高等学校计算机类国家级特色专业系列规划教材

Python程序设计

——计算思维视角

陈杰华 编著

清华大学出版社

高等学校计算机类国家级特色专业系列规划教材

Python 程序设计
——计算思维视角

陈杰华　编著

清华大学出版社
北京

内 容 简 介

本书共分两部分，第一部分包括计算思维与 Python 简介、算法、数据与计算、流程控制、函数、模块、数据文件、面向对象编程、异常处理、图形界面设计和绘制曲线，为方便教学，每章最后均附有简答题和编程题；第二部分给出了 7 个实验，即数据与计算、流程控制、函数、数据文件、面向对象编程、图形界面设计和绘制曲线。

本书按课程教学模式来组织内容，既适合教师授课，也适合学生自学；同时，本书面向应用需求，教学内容先进，尽力帮助学生理解并掌握计算思维和编程技术。

本书内容丰富、图文并茂，讲解简明易懂、循序渐进、深入浅出，可作为高等学校非计算机专业学生学习“Python 程序设计”课程的教材，也可作为初学者、Python 爱好者的辅助学习资料。

图书在版编目(CIP)数据

Python 程序设计：计算思维视角/陈杰华编著. —北京：清华大学出版社，2018
(高等学校计算机类国家级特色专业系列规划教材)
ISBN 978-7-302-51341-4

Ⅰ. ①P…　Ⅱ. ①陈…　Ⅲ. ①软件工具－程序设计－高等学校－教材　Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2018)第 229143 号

责任编辑：汪汉友
封面设计：傅瑞学
责任校对：胡伟民
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　**邮　　编：**100084
社 总 机：010-62770175　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市君旺印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张：**22　**字　　数：**540 千字
版　　次：2018 年 12 月第 1 版　**印　　次：**2018 年 12 月第1次印刷
定　　价：49.00 元

产品编号：079062-01

前　　言

Python 语言是由荷兰的 Guido van Rossum 于 1989 年发明的，是一种适合教学的通用语言，其符号体系与英语的描述方式完全一致，具有极佳的阅读性。初学者在编程时只需专注求解问题本身，而不必花过多时间和精力在语言的语法细节和具体实现上，因此在培养计算思维方面具有明显优势。计算思维包括合理的问题抽象、精准的程序描述和自动化的机器实现。在丰富的数据类型、完备的计算能力和多种运行模式这 3 个方面，Python 语言均提供了很大支持。伴随着多媒体技术、互联网＋、大数据、云计算、物联网、人工智能等高新技术的发展，Python 语言也与时俱进地迅猛发展起来。

1. 本书特点

本书主要特点体现在如下两个方面。

（1）适合师生教学。本书参照教育部 2015 年 11 月制定的《大学计算机基础课程教学基本要求》进行编写，教学内容组织合理、条理清晰，在基础知识部分，为了便于学习，每章均配有简答题和编程题；实验部分是程序设计和计算思维方面的机器实现，利用与书配套的电子课件、习题答案、实验素材等教学资源，更便于教学和上机实验。

（2）章节结构合理。本书按照 Python 语言和计算思维概念安排各个章节，易于读者理解。每章均按照语法描述、示例讲解和编程实现的逻辑来介绍程序设计，有利于学生对照学习，提高学习效率。本书采用循序渐进的学习模式，适合初、中级读者掌握编程方法，并最终能够编写中小规模的实用程序。

2. 章节安排

本书按课程教学方式来组织内容，因此适合教师授课，也适合学生阅读。全书共分两部分，第一部分包括 11 章，具体内容安排如下。

第 1 章内容包括程序、程序设计与操作步骤、计算机语言与分类、程序翻译等与计算机求解问题相关的知识，问题求解、设计系统和人类行为理解这 3 个计算思维的应用领域，Python 语言的概貌、下载与安装、帮助信息系统以及 5 种运行模式。

第 2 章内容包括面向过程和面向对象的程序设计方法；算法概念以及如交换两个变量内容、取绝对值、阶加、阶乘、求最大公约数、求斐波那契数列、判断素数等常用算法，数值计算、穷举算法、查找算法、排序算法等综合算法方面的运用，以及迭代、递推、递归等。

第 3 章内容包括 Python 的输入输出操作、编码风格、简单程序、组合符号、数据类型概念、数字、字符串、布尔数据、列表、元组、字典、集合等数据类型及其运用。

第 4 章内容包括分支选择和循环控制的语句与编程，包含 if、while、for、continue、break、pass 语句和 range()函数，以及列表处理、查找、排序和字符串处理编程案例。

第 5 章内容包括自定义函数及其调用、嵌套调用、返回列表、形式参数与实在参数、全局变量与局部变量、lambda 函数、递归函数等。

第 6 章内容包括 math、cmath、decimal、fractions、random、time、datetime、calendar、time、os、sys 等模块和运用，以及如何自定义模块和包。

第 7 章内容包括数据文件的概念，文件的打开与关闭，如何读写 ASCII 文件和二进制文件，文件读写操作的 struct 模块、fileinput 模块和 codecs 模块及其运用。

第 8 章内容包括对象、类、继承、多态等面向对象编程的基本概念，类的定义和引用，继承、多态和重载的实现方法。

第 9 章内容包括 Python 语言的异常处理机制，如何抛出和捕捉异常，在 Python 程序中如何处理异常以及如何自定义异常类。

第 10 章内容包括 Python 图形开发库的介绍，布局的管理按钮、输入框、框架、标签、列表框、菜单、滚动条、文本框、滑动杆、面板、对话框、消息框等图形界面对象的使用，以及事件和事件处理程序的编写。

第 11 章内容包括如何使用画布(Canvas)组件绘制直线、矩形、多边形、圆弧、椭圆，显示位图、图像与文本，控制与变换图形，用 Python 内置的海龟程序和海龟绘图，生成分形(Fractal)图形，以及两种显示字体的方法。

第二部分包括 7 个实验，即数据与计算、流程控制、函数、数据文件、面向对象编程、图形界面设计和绘制曲线。

3. 教学建议

课堂教学和上机实验的建议学时与安排见下表。

章节(48 学时)	课堂教学(32 学时)	上机实验(16 学时)
第 1 章　计算思维与 Python 简介	2	
第 2 章　算法	2	
第 3 章　数据与计算	4	3
第 4 章　流程控制	5	3
第 5 章　函数	3	2
第 6 章　模块	3	
第 7 章　数据文件	3	2
第 8 章　面向对象编程	2	2
第 9 章　异常处理	2	
第 10 章　图形界面设计	3	2
第 11 章　绘制曲线	3	2

4. 图例说明

由于 Python 使用了 5 种运行模式，为了让读者方便区分并获取精确内容，已对全书中的图例进行统一剪裁，以尽量减少冗余信息。当然，若图例本身完全不能剪裁，则会保留原图不变。

(1) Windows 窗口：剪裁外框中的空白区域。

(2) 程序运行结果：剪裁全部外框，只保留上边框至 RESTART 标记，以及右边框至后续 23 个等号处。

(3) IDLE 交互窗口：剪裁全部外框。

(4) IDLE 命令行窗口:剪裁全部外框,保留标题栏。

(5) Windows 命令提示符窗口:剪裁全部外框,保留标题栏。

(6) 图形界面设计和绘制曲线两章:剪裁外框中的空白区域。

本书由陈杰华制定全书的整体框架并负责统稿工作、编写主要内容,四川大学计算机教学中心的孟宏源和戴丽娟参与编写部分内容、资料整理、代码调试、图片制作、结构设计等工作。在本书编写过程中,得到四川大学教务处、计算机学院和计算机教学中心的领导和老师的许多帮助,在此表示由衷感谢。在本书编写过程中,翻阅了大量文献,很受启发,在此向所有前辈和学者表示由衷的敬意和感谢。最后感谢清华大学出版社相关编校人员为本书出版所做工作。

本书配套技术问题、索要例题与习题源程序文件、电子教案和教学素材,可从清华大学出版社本书页面下载,也可以发送电子邮件联系我们寻求帮助。编者电子邮件地址为 cjh028@126.com 和 chenjiehua@scu.edu.cn。

由于作者水平有限,书中难免有不足之处,恳请广大前辈、学者和读者批评指正。

陈杰华

2018 年 10 月

目　录

第一部分　基础知识

第 1 章　计算思维与 Python 语言 …… 3
1.1　程序设计 …… 3
1.1.1　程序与计算机程序 …… 3
1.1.2　程序设计步骤 …… 5
1.1.3　程序设计语言 …… 6
1.1.4　高级语言分类 …… 7
1.2　计算思维 …… 8
1.2.1　计算思维概念 …… 9
1.2.2　计算思维特征 …… 11
1.3　Python 简介 …… 12
1.3.1　Python 优点 …… 12
1.3.2　Python 缺点 …… 14
1.3.3　Python 主要应用 …… 15
1.4　Python 运行环境 …… 16
1.4.1　Python 下载与安装 …… 16
1.4.2　Python 帮助信息 …… 19
1.4.3　Python 文件夹结构 …… 21
1.4.4　Python 运行模式 …… 21
习题 1 …… 27

第 2 章　算法 …… 29
2.1　程序设计方法 …… 29
2.1.1　结构化程序设计方法 …… 29
2.1.2　面向对象程序设计方法 …… 30
2.2　算法 …… 32
2.2.1　求解问题方式 …… 32
2.2.2　算法概念 …… 32
2.2.3　算法特征 …… 32
2.3　算法表示 …… 33
2.3.1　使用自然语言描述算法 …… 33
2.3.2　使用传统流程图描述算法 …… 34

2.3.3 使用 N-S 图描述算法 …… 36
2.3.4 使用伪代码描述算法 …… 37
2.4 常用算法介绍 …… 37
2.4.1 简单算法 …… 37
2.4.2 阶乘算法 …… 38
2.4.3 求斐波那契数算法 …… 38
2.4.4 求最大公约数算法 …… 39
2.4.5 判断素数算法 …… 40
2.5 综合算法介绍 …… 40
2.5.1 数值计算 …… 40
2.5.2 穷举算法 …… 41
2.5.3 查找算法 …… 43
2.5.4 排序算法 …… 44
2.6 迭代、递推和递归 …… 46
2.6.1 迭代 …… 46
2.6.2 递推 …… 47
2.6.3 递归 …… 48
习题 2 …… 48

第 3 章 数据与计算 …… 50
3.1 输入输出 …… 50
3.1.1 输入数据 …… 50
3.1.2 输出数据 …… 51
3.2 编码风格与简单程序 …… 53
3.2.1 编码风格 …… 53
3.2.2 简单程序 …… 53
3.3 组合符号 …… 54
3.3.1 标识符 …… 54
3.3.2 关键字 …… 55
3.3.3 预定义标识符 …… 56
3.3.4 命名规则 …… 57
3.4 数据类型 …… 57
3.4.1 数据类型及其分类 …… 57
3.4.2 常量和变量 …… 58
3.5 数字数据 …… 58
3.5.1 整型数据 …… 58
3.5.2 实型数据 …… 61
3.5.3 分数型数据 …… 62
3.5.4 复数型数据 …… 62

3.6 字符串型数据…… 63
3.6.1 字符串常量 …… 63
3.6.2 转义字符 …… 63
3.6.3 字符串测试函数 …… 64
3.6.4 字符串运算符 …… 65
3.6.5 字符串内置函数 …… 66
3.7 布尔型数据…… 68
3.7.1 关系运算 …… 68
3.7.2 布尔常量 …… 69
3.7.3 布尔运算 …… 69
3.8 序列数据…… 70
3.8.1 列表 …… 70
3.8.2 元组 …… 74
3.8.3 字典 …… 74
3.8.4 集合 …… 76
习题 3 …… 77

第 4 章 流程控制 …… 80
4.1 简单程序与流程控制…… 80
4.1.1 简单程序 …… 80
4.1.2 流程控制语句 …… 81
4.1.3 测试条件 …… 81
4.2 分支选择…… 81
4.2.1 单分支选择 …… 82
4.2.2 双分支选择 …… 83
4.2.3 多分支选择 …… 84
4.3 循环控制…… 86
4.3.1 while 语句 …… 86
4.3.2 range()函数…… 89
4.3.3 for 语句…… 89
4.3.4 循环嵌套 …… 94
4.3.5 continue、break 和 pass 语句 …… 98
4.4 列表处理 …… 101
4.4.1 一维列表…… 102
4.4.2 二维列表…… 107
4.5 查找与排序 …… 114
4.5.1 折半查找…… 114
4.5.2 排序…… 115
4.6 字符串处理 …… 118

4.6.1　单个字符串 …… 118
4.6.2　多个字符串 …… 122
习题 4 …… 123

第 5 章　函数 …… 125
5.1　函数定义与调用 …… 125
5.1.1　函数定义与调用 …… 125
5.1.2　嵌套调用 …… 128
5.1.3　返回值类型与函数类型 …… 129
5.1.4　返回列表 …… 130
5.2　形式参数与实在参数 …… 130
5.2.1　简单变量作为实参 …… 131
5.2.2　一维列表作为实参 …… 137
5.2.3　二维列表作为实参 …… 139
5.2.4　可变参数 …… 139
5.3　变量的作用域 …… 140
5.3.1　全局变量与局部变量 …… 140
5.3.2　global 语句 …… 141
5.3.3　变量同名 …… 143
5.4　匿名函数 …… 144
5.4.1　lambda 函数 …… 144
5.4.2　程序示例 …… 144
5.5　递归函数 …… 145
5.5.1　递归函数及其调用 …… 145
5.5.2　程序示例 …… 145
习题 5 …… 151

第 6 章　模块 …… 152
6.1　模块 …… 152
6.1.1　导入模块 …… 152
6.1.2　导入模块成员 …… 153
6.1.3　模块搜索路径 …… 154
6.2　数值类模块 …… 155
6.2.1　math 模块 …… 155
6.2.2　cmath 模块 …… 157
6.2.3　decimal 模块 …… 157
6.2.4　fractions 模块 …… 159
6.3　random 模块 …… 161
6.3.1　常用函数 …… 161

6.3.2 程序示例 …… 163
6.4 时间类模块 …… 165
6.4.1 time 模块 …… 167
6.4.2 datetime 模块 …… 171
6.4.3 calendar 模块 …… 172
6.5 os 模块 …… 174
6.5.1 常用函数 …… 174
6.5.2 程序示例 …… 176
6.6 sys 模块 …… 178
6.6.1 常用函数 …… 178
6.6.2 命令行参数 …… 179
6.7 自定义模块 …… 180
6.7.1 主模块 …… 181
6.7.2 自定义模块示例 …… 182
6.7.3 Python 编译文件 …… 183
6.8 自定义包 …… 183
6.8.1 包与模块的组织结构 …… 183
6.8.2 包与模块的导入 …… 184
6.8.3 自定义包示例 …… 185
习题 6 …… 187

第 7 章 数据文件 …… 189
7.1 文件概述 …… 189
7.1.1 引言 …… 189
7.1.2 文件分类 …… 189
7.2 打开文件与关闭文件 …… 191
7.2.1 打开文件 …… 191
7.2.2 关闭文件 …… 193
7.3 读写文本文件 …… 193
7.3.1 读取文件函数 …… 193
7.3.2 读取文本文件 …… 194
7.3.3 写入文本文件 …… 198
7.4 读写二进制文件 …… 200
7.4.1 将字符串转换为字节数据 …… 201
7.4.2 将字节数据转换为字符串 …… 201
7.4.3 读写二进制文件 …… 202
7.5 struct 模块 …… 202
7.5.1 pack()、unpack()和 calcsize()函数 …… 203
7.5.2 程序示例 …… 204

7.6 fileinput 模块 …… 206
7.6.1 fileinput 模块 …… 206
7.6.2 程序示例 …… 206
7.7 codecs 模块 …… 208
7.7.1 读取文本文件 …… 209
7.7.2 写入文本文件 …… 209
习题 7 …… 210

第 8 章 面向对象编程 …… 212
8.1 面向对象编程基础 …… 212
8.1.1 对象与类 …… 212
8.1.2 对象特征 …… 213
8.1.3 继承 …… 213
8.1.4 多态性与重载 …… 214
8.2 类的定义和引用 …… 215
8.2.1 类的构成 …… 215
8.2.2 类的定义与引用 …… 215
8.2.3 构造函数和析构函数 …… 217
8.2.4 实例变量 …… 219
8.2.5 私有成员与公有成员 …… 220
8.2.6 公有方法与私有方法 …… 221
8.3 继承 …… 222
8.3.1 单继承 …… 222
8.3.2 多继承 …… 223
8.3.3 方法重写 …… 224
8.4 多态与运算符重载 …… 224
8.4.1 多态 …… 224
8.4.2 运算符重载 …… 225
习题 8 …… 226

第 9 章 异常处理 …… 229
9.1 程序错误及其处理 …… 229
9.1.1 程序错误类型 …… 229
9.1.2 程序运行错误处理方法 …… 232
9.2 标准异常 …… 233
9.2.1 标准异常 …… 233
9.2.2 标准异常示例 …… 234
9.3 抛出异常和捕捉异常 …… 235
9.3.1 抛出异常 …… 235

9.3.2 捕捉异常…………………………………………………………… 237
9.4 断言 ……………………………………………………………………… 240
9.4.1 断言概念………………………………………………………… 240
9.4.2 assert 语句 ……………………………………………………… 240
9.5 自定义异常类 ………………………………………………………… 242
9.5.1 引言……………………………………………………………… 242
9.5.2 程序示例………………………………………………………… 242
习题 9 ………………………………………………………………………… 243

第 10 章 图形界面设计 ………………………………………………… 245
10.1 Python 图形界面设计 ……………………………………………… 245
10.1.1 Python 图形开发库 ………………………………………… 245
10.1.2 Tkinter 的常用组件与标准属性 …………………………… 246
10.1.3 创建窗口…………………………………………………… 246
10.2 布局管理…………………………………………………………… 247
10.2.1 pack 布局的管理 …………………………………………… 247
10.2.2 grid 布局的管理……………………………………………… 249
10.2.3 place 布局的管理 …………………………………………… 251
10.3 Tkinter 的常用组件 ………………………………………………… 252
10.3.1 Label 组件 ………………………………………………… 252
10.3.2 Button 组件 ………………………………………………… 253
10.3.3 Entry 和 Text 组件 ………………………………………… 255
10.3.4 Listbox 组件 ……………………………………………… 257
10.3.5 Radiobutton 和 Checkbutton 组件 ………………………… 259
10.3.6 Frame 与 LabelFrame 组件 ………………………………… 261
10.3.7 Scrollbar 组件 ……………………………………………… 262
10.3.8 Menu 组件 ………………………………………………… 264
10.3.9 对话框……………………………………………………… 266
10.4 事件处理…………………………………………………………… 271
10.4.1 事件类型…………………………………………………… 271
10.4.2 事件绑定…………………………………………………… 273
10.4.3 键盘事件…………………………………………………… 275
习题 10 ……………………………………………………………………… 276

第 11 章 绘制曲线 ……………………………………………………… 278
11.1 Canvas 组件 ………………………………………………………… 278
11.1.1 Canvas 对象及其通用属性 ………………………………… 278
11.1.2 屏幕坐标…………………………………………………… 279
11.2 绘制图形…………………………………………………………… 279

11.2.1 绘制直线、矩形和多边形 …… 279
11.2.2 绘制圆弧和椭圆 …… 282
11.3 显示位图、图像与文本 …… 285
11.3.1 显示位图 …… 285
11.3.2 显示图像 …… 286
11.3.3 显示文本 …… 287
11.4 控制图形 …… 288
11.4.1 删除图形 …… 288
11.4.2 移动图形 …… 289
11.4.3 位置坐标 …… 289
11.4.4 缩放图形 …… 290
11.4.5 绘制函数图形 …… 292
11.5 体验内置的 turtle 演示程序 …… 293
11.5.1 利用 IDLE 内置程序 …… 293
11.5.2 利用安装文件夹中的演示程序 …… 294
11.6 turtle 绘图 …… 296
11.6.1 turtle 模块 …… 296
11.6.2 应用案例 …… 297
11.7 分形图形 …… 301
11.7.1 Koch 曲线 …… 301
11.7.2 Hilbert 曲线 …… 302
11.7.3 分形树 …… 304
11.8 显示字体 …… 306
11.8.1 通过元组显示字体 …… 306
11.8.2 通过 Font 对象显示字体 …… 307
习题 11 …… 308

第二部分　实　　验

实验Ⅰ　数据与计算 …… 313

实验Ⅱ　流程控制 …… 316

实验Ⅲ　函数 …… 319

实验Ⅳ　数据文件 …… 323

实验Ⅴ　面向对象编程 …… 325

实验Ⅵ　图形界面设计……………………………………………………………… 329

实验Ⅶ　绘制曲线………………………………………………………………………… 333

参考文献…………………………………………………………………………………… 336

第一部分　基础知识

第 1 章　计算思维与 Python 语言

计算机作为计算工具，主要用途就是求解各种应用问题。求解问题一定需要各种程序设计技术和合适的程序描述语言，常用的程序语言有 Java、C、JavaScript、Python 等。本章首先介绍计算机程序、程序设计及其操作步骤、计算机语言及其分类、程序翻译等与计算机求解问题相关的许多知识，其后介绍了计算机科学家进行问题求解的方法(即计算思维)，详细说明了计算思维的 3 个应用领域：问题求解、设计系统和人类行为理解，最后介绍运用计算机实现程序和计算思维的工具——Python 语言，包括 Python 语言的概貌、下载与安装、帮助信息系统和 5 种运行模式。

通过本章学习，读者可在 Python 环境中进行操作并编写简单程序。

1.1　程序设计

程序设计是软件开发的基础，应当遵循软件开发的标准与规范。本节基于软件工程实践的原则和方法，详细介绍了计算机程序的构造方法、计算机语言与翻译程序，并通过 3 个 Python 程序讲述编程的思想与方法。

1.1.1　程序与计算机程序

程序一词自古有之，过去用于描述完成一件事情的操作过程，例如一次宗教仪式、一次开会议程、一趟旅行安排等。总之，程序就是为完成特定任务而安排的一系列操作过程及其描述。

那么，在计算机领域中程序的含义是什么？什么才能被称为计算机程序呢？这种程序与现实生活中的程序又有什么相同点与不同点呢？例如，给程序员一台计算机，他将如何借助这台计算机来求解一个一元二次方程的两个解？

人们使用计算机，就是要利用它处理各种现实问题。但是，计算机只是一种计算工具，本身并没有自行求解问题的能力，如同算盘一样，都需要人为进行控制才能实现计算。不过，计算机的工作原理与算盘完全不同，这种不同是计算机求解问题需要程序员写出代码形式的操作步骤。先来看看计算机是如何求解一元二次方程的。

【例 1-1】　一元二次方程的计算机求解过程。

用计算机求解一元二次方程需要安排的操作步骤如下：

步骤 1，将一元二次方程的 3 个系数 a、b 和 c 输入计算机；

步骤 2，用公式计算方程的第 1 个根；

步骤 3，用公式计算方程的第 2 个根；

步骤 4，显示两个根；

步骤 5，结束。

现在，来观察计算机程序的执行过程。计算机在执行预先由程序员安排好的计算操作

时，将完全按照程序员指定的操作指令(即程序)去做，例如操作指令是“相加”，它不能进行“相减”，更不能计算错误。在计算机中，操作指令对应于计算机能够执行的一个基本动作。为求解一个计算问题，程序员会让计算机按照特定操作顺序完成一系列的指令，这一系列指令的集合就是计算机程序。

下面写出求解一元二次方程的 Python 程序。

```
#导入数学模块 math
import math
#提示输入数据
print("键盘输入数据:")
a=float(input("a="))
b=float(input("b="))
c=float(input("c="))
#计算方程的两个根
x1=(-b-math.sqrt(b*b-4*a*c))/2/a
x2=(-b+math.sqrt(b*b-4*a*c))/2/a
print("显示计算结果:")
print("方程的第 1 个根:",x1)
print("方程的第 2 个根:",x2)
```

本例中的第 1 行是由符号#开始的注释行；第 2 行用于导入数学模块 math，以便后续语句能够调用开平方根函数 sqrt()，否则 Python 解释器将因不能获取所需函数而运行失败；第 5～7 行调用 input()函数实现键盘输入，这里的 float()函数实现从字符串数据到实型数据的转换；第 9～10 行利用数学公式分别计算方程的两个根；第 12～13 行显示两个根并结束程序。

本例中的第 1 行使用符号#是在 Python 程序中表示注释，本身没有执行意义，但合理的注释则能够增加程序的可读性。使用符号#的一种方式是在独立的一行中进行书写，另一种方式是在一条语句的最后进行书写。另外，直接使用 print()函数显示信息也可以增加程序的阅读效果。

运行结果如图 1-1 所示。

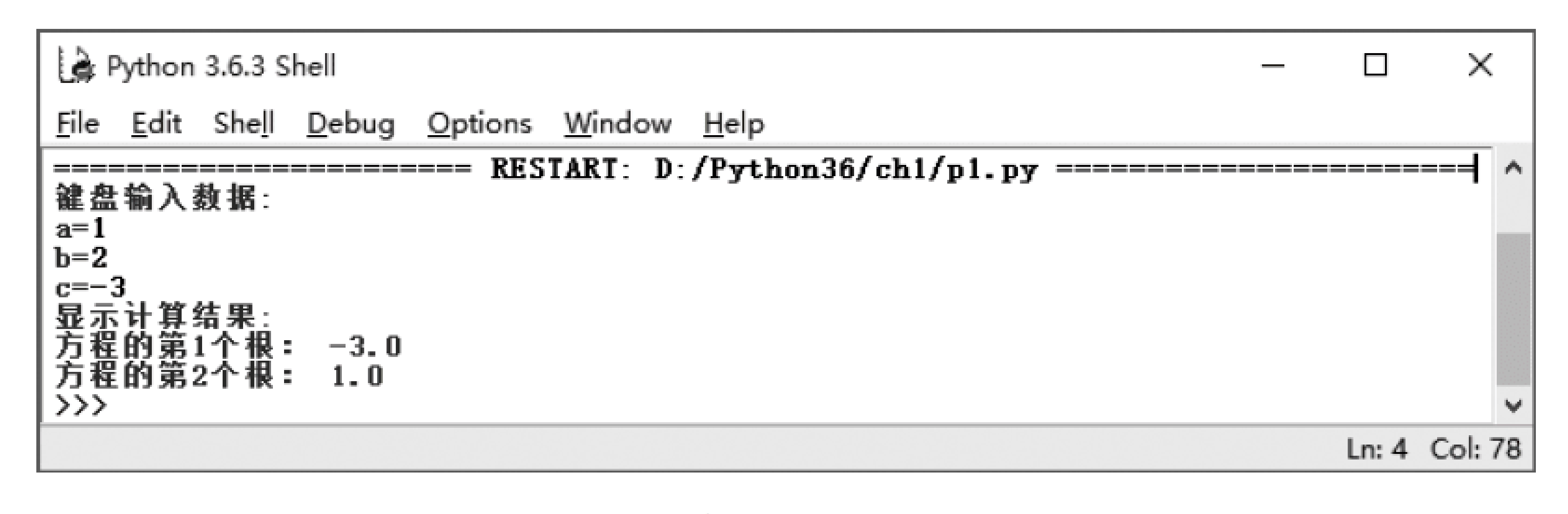


图 1-1　求解一元二次方程

如图 1-1 所示，界面中以 23 个符号=导引的是当前正在运行的程序，棕色符号>>>(连续的 3 个大于符号)是表示 IDLE 交互环境的提示符，黑色文字表示用户输入的数据，蓝色文字表示程序运行中的显示信息。

注：本书中的全部例题程序，均以统一方式指定其对应的文件名称：由字符串 ch 开头，

后面的第 1 个数字表示章的序号，第 2 个数字表示例题程序的序号(两位数字)，且二者使用下画线连接，后缀必须为.py 以表示 Python 源程序，例如文件名 ch1_01.py 表示第 1 章的第 1 个程序。

从上述关于求解一元二次方程的过程中，可以发现程序中的操作描述就是控制计算机自动执行的，而程序设计就是为控制计算机而设计程序的全部过程。

1.1.2 程序设计步骤

要用计算机求解一个计算问题，就要先描述解决该计算问题的过程，这是软件构造活动中的重要组成部分之一。程序设计就是以一种程序设计语言为描述工具，给出相应语句序列的过程，其目标就是使用计算机来求解相应的计算问题。

由于求解问题本身的复杂性和构造程序的协同制造模式，通常不会直接编写程序，而是要将其分解成若干个步骤。一般而言，程序设计的基本步骤包括分析问题、设计算法、编写程序、运行程序、分析结果、编写文档等，下面分别进行说明。

1. 分析问题

分析问题就是对于计算问题要进行认真分析，研究已知条件，分析最后应该得到的结果，找出求解问题的规律，选择合理的解题思想。其中包含许多关于问题求解的抽象，主要目标就是要让计算机能够自动化地运行合理的程序并最终得到问题求解。

2. 设计算法

设计算法就是设计解题过程的具体方法和相关操作步骤，如果有多种算法均可以实现问题求解，则可以从中选择最佳算法。评价算法的标准除正确性外，还有可读性，而保证可读性则要求编程过程必须严格遵循框图模式、语句模式、编码风格等方面的规范。

3. 编写程序

目前，常用的程序设计语言有数十种，分别用于处理相关应用方面的计算问题，例如 Java 语言适于网络编程，Python 适于编程教学。在编写程序前，程序员要根据实际应用要求选择合适的程序设计语言，并按照算法设计思想来编写程序，并对源程序进行编辑、翻译、连接等操作。

4. 运行程序

运行程序即利用特定的计算机执行系统(软件与硬件)来运行程序，通常需要提供合理的输入数据，并要求最终能够得到正确的运行结果。例如，运行 Python 程序需要合理的字长(属于硬件)和相关的版本(属于软件)。

5. 分析结果

通过运行程序得到运行结果后，并不一定表示程序完全正确，所以要对结果进行分析，检查是否正确，若不正确，则要对程序进行进一步的修改并调试，直到完全正确为止。

6. 编写文档

由于程序最终都是要提供给用户使用的。如同销售商品需要说明书，提供给用户所用的程序，同时必须向用户提供说明书和操作指南，这些说明书和操作指南都是程序开发者必须提供给用户的，主要内容包括程序功能、运行环境要求、程序的安装与启动过程、输入数据要求、使用注意事项等。另外，由于程序调试和维护需要，也应该在程序清单中添加许多注释方面的信息。

1.1.3 程序设计语言

任何程序是由一系列符号按特定规则组成的，而程序设计语言就是编程过程中的描述工具，如同国内作家用汉语创作文学作品一样。

自 20 世纪 50 年代末期以来，计算机行业内出现数千种程序设计语言，但目前只有数十种语言得到了广泛应用。从历史发展历程来看，可以将程序设计语言分为四个时代。

1. 第一代——机器语言

机器语言就是由二进制 0、1 代码形式构成的指令，不同的计算机系统具有不同的指令系统。例如，某台计算机中，表示相加运算的指令代码就是 01001，被加数和加数也是 0、1 代码形式的代码。很明显，记忆和使用二进制代码是非常困难的。另外，使用机器语言编写程序没有通用性，即换另一台式计算机后就不能运行原来的程序，并且程序的修改与维护也极不方便。所以编程效率极低，目前这种语言已经基本上被其他更"高级"的计算机语言代替了。

无可否认，机器语言中的指令可由计算机直接识别和执行，运行速度快，且可以实现对硬件的直接控制，所以在实时控制、自动化设备、机器人技术、游戏设计等领域还需要机器语言。

2. 第二代——汇编语言

汇编语言中的语句就是机器指令的符号化形式，即使用十进制数据取代二进制数据，使用英文单词(缩写词)来取代二进制形式的机器指令。例如二进制数据 1101 可改写成十进制数字 13，指令代码 01001 可改写成相加运算的 ADD。汇编语言的语句与机器语言的指令存在着一一对应的关系，所以汇编语言也同样存在难学、难用、易出错、维护困难、功能弱等缺点。但是，汇编语言也有其优点，例如可以直接访问硬件、比机器语言的编程效率高、易读等。还有很重要的一点，计算机不能直接识别汇编语言中的符号化语句，这就要求必须将汇编语言源程序翻译成二进制形式的机器指令(或称为目标程序)。这种翻译称为"汇编"，所用翻译软件称为"汇编程序"，如图 1-2 所示。

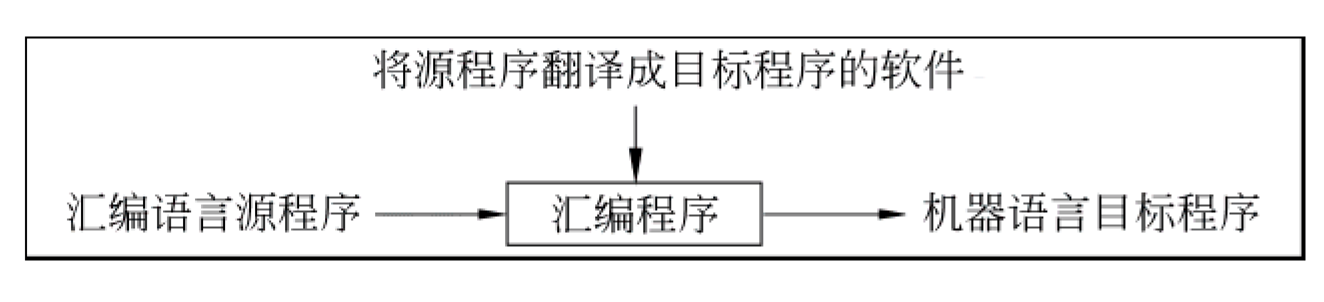


图 1-2 汇编过程

3. 第三代——高级语言

为解决机器语言和汇编语言不能通用的缺陷，1955 后开始出现了 BASIC、FORTRAN 等高级语言，它们均是面向用户的、独立于计算机硬件的编程语言。首先，高级语言的书写形式接近于数学语言和英语，思维形式与传统人类求解问题方法比较接近；其次，高级语言中的一条语句可以代替数条甚至数十条汇编语言中的指令，所以高级语言具有通用性强、应用广泛、可读性好、功能强等特点。

不过，计算机不能直接识别高级语言中的语句，所以需要翻译成二进制形式的机器指令。这种翻译通常分为称为"编译"和"解释"两种方式，所用的翻译软件称为"编译程序"和"解释程序"。

(1) 编译方式是把源程序中的每条语句都编译成机器语言后，并保存为二进制的目标

文件，这样运行时计算机可以直接以机器语言形式的目标程序来运行程序，运行速度很快，如图 1-3 所示。

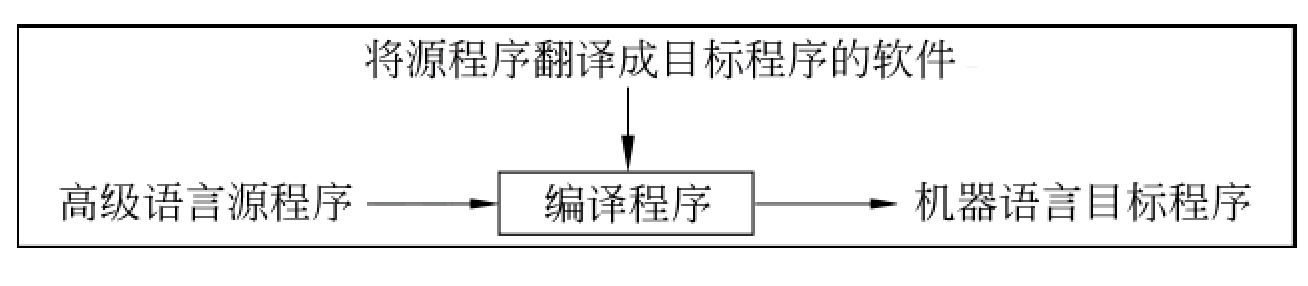


图 1-3　编译过程

(2) 解释方式则是只在执行程序时才一条一条的解释成机器语言形式的指令，并由计算机执行，所以运行速度较慢，如图 1-4 所示。

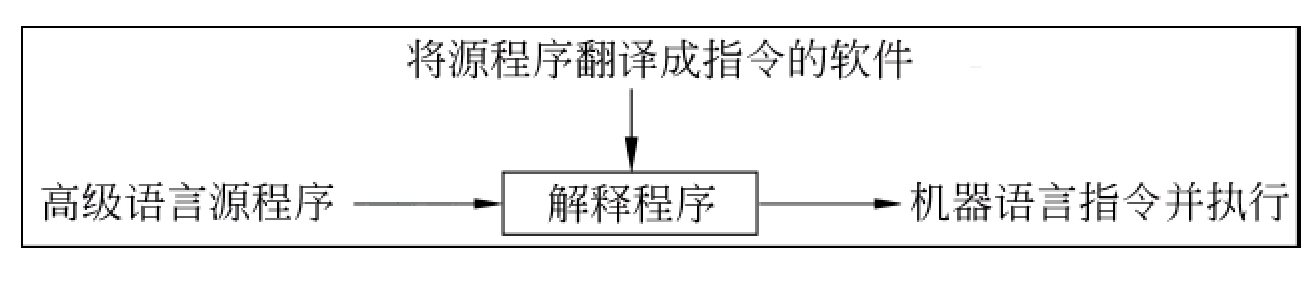


图 1-4　解释过程

这两种翻译方式类似于汉语中的笔译与口译，由于口译过程中可以不涉及后一句的翻译问题。所以，从翻译处理角度来看，口译比笔译简单。Python 语言使用的是解释方式，从而为使用者提供较好的操作环境。

4. 第四代——非过程化语言

使用非过程化语言编程时只需告诉计算机“做什么”而不是“怎样做”，即不需要描述算法具体实现的细节，其中的两个典型应用是数据库查询和应用程序生成器。属于这类语言的有 System Z、PowerBuilder、FOCUS 等。例如，PowerBuilder 语言是 Sybase 公司研制的一种新型快速开发工具，使用客户机—服务器结构，基于 Windows 环境的集成化开发工具。主要特点包括可视化与多特性的开发工具、面向对象技术、支持高效的复杂应用程序、数据库的连接能力、查询、报表和图形功能等。

Python 具有非过程化语言的描述能力，并提供完整的对象编程技术。

1.1.4　高级语言分类

高级语言种类繁多，可以从应用特点和对客观描述两个方面进行分类。

1. 从应用角度分类

从应用角度来看，高级语言可以分为 3 类：基础语言、专用语言和结构化语言。

(1) 基础语言。基础语言也称为通用语言。这种高级语言历史悠久，目前有大量的软件库，为众多用户所熟悉和使用。属于这类语言的有 FORTRAN、BASIC、COBOL、ALGOL 等。例如，BASIC 语言是 20 世纪 50 年代中期为适应分时系统而研制的一种人机交互式的程序设计语言，可用于一般的程序设计教学、数值计算、数据处理、自动控制等。目前，微软公司推出的一种新型版本是 Visual Basic。

(2) 专用语言。专用语言是为某种特殊应用而专门设计的语言，通常具有特殊的应用需求和语法形式。一般而言，这种语言的应用范围狭窄，移植性和可维护性都较差。属于这类语言的有 LISP、APL、Forth 等。例如，Forth 语言是 20 世纪 60 年代出现的基于堆栈处理、人机交互等功能的编程语言，主要应用于软件代码超过数千行规模的嵌入式系统。该语

言被广泛应用于军事、航天、航空、工业自动化、工作站、图形、仪器仪表、天文等领域，目前已研制出以 Forth 为体系结构的处理器芯片。

(3) 结构化语言。从 20 世纪 70 年代初期开始，结构化程序设计思想就日益为人们所接受、欣赏和使用，其间先后出现了许多结构化语言。这些结构化语言支持结构化的控制结构，具有完备的关于过程结构和数据结构的描述能力，属于这类语言的有 Pascal、C、Ada 等语言。例如，C 语言具有使用方便、应用面广、功能强大、描述能力强、运算符与数据类型丰富、易移植、编译质量高、代码优化等优点。同时，C 语言还具有机器语言的许多优点，例如允许直接访问物理地址、进行二进制的位操作、直接控制硬件等。

Python 同时具有基础语言和结构化语言的主要特征。

2. 从描述方式分类

从描述客观系统来看，程序设计语言可以分为两类：面向过程语言和面向对象语言。

(1) 面向过程语言。面向过程语言是以“程序＝数据结构＋算法”的程序设计范式构成的编程语言。属于这类语言的有 BASIC、FORTRAN、C、Pascal 等。面向过程的核心是自顶向下、逐步求精的功能分解方法，在进行软件设计和编码时，一般需要进行如下 4 个步骤：

① 将实际应用问题分解成若干个功能模块；

② 根据功能模块的要求设计一系列用于存储数据的数据结构；

③ 绘制许多求解功能模块的算法框图；

④ 编写对相关数据结构实施操作的过程、函数、子程序等。

最终的程序就是由这些过程、函数和子程序构成的。显然，面向过程语言是将数据结构和过程实现作为两个实体分别对待的，其着重点在于过程实现，而不是对数据结构进行描述。程序员在设计程序时，首先要考虑的就是如何进行功能分解，在每一个过程实现中又要着重安排程序的操作序列；另一方面，程序员在编程时又必须经常考虑对应的数据结构，因为操作毕竟是作用于特定数据结构上的。

(2) 面向对象语言。面向对象语言是以“程序＝对象＋消息”的程序设计范式构成的编程语言，属于这类语言的有 Visual Basic、C++、Java、Delphi、Python 等。例如，Java 语言是一个简单的、面向对象的、分布式的、解释的、健壮的、安全的、独立于平台的、可移植的、高性能的、多线程的、动态的程序设计语言。

要解决面向过程程序设计技术的可扩展性差、维护代价高、抽象程度强等缺点，只有使用面向对象或基于对象的程序设计技术。面向对象程序设计技术是通过增加软件的可扩充性和可重用性来提高程序员的编程能力，它的最大优点是软件具有可重用性，这种新技术更接近人的思维活动。人们利用面向对象思想进行程序设计时，可以很大程度地提高编程效率，并减少软件维护的开销。当人们对软件系统的要求发生变化时，并不需要程序员做大量的修改工作。

Python 作为现代语言，同时具有面向过程编程和面向对象编程的两种技术。

1.2 计算思维

在介绍程序、程序设计、计算机语言等概念后，下面就可以学习计算机科学家们是如何思考并编写程序的，这就是计算思维。

1.2.1　计算思维概念

计算思维是指从具体的算法设计规范入手，通过算法过程的构造与实施来解决给定问题的一种思维方法。它以设计和构造为特征，以计算机学科为代表。计算思维是运用计算机科学的基础概念去求解问题、设计系统和理解人类行为的一系列思维活动。

从上述定义中，可以清楚发现计算思维的 3 个应用领域，即问题求解、系统设计和人类行为理解。下面只能简单说明，具体细节和详细描述在后续章节中进行展开。

1. 问题求解

计算思维是指将问题求解的过程用“程序化”或“自动化”的方式表示出来。程序员在面对计算问题时，可依据已有的知识，提出问题求解方案，并用算法进行描述，最终由机器执行程序来检验结果是否合理。例如，乘车分段收费问题就是人们在日常生活中感受到的问题，读者可根据自己对火车分段收费操作的理解，写出数学关系式(例如分段函数)，并利用多分支选择结构来进行算法描述，最后由机器实现并得到结果。下面是一个用计算机很容易求解的问题。

【例 1-2】　找出整数 281631327 中的全部因数。

显然，用过去学过的数学知识和计算手段进行此类问题求解是非常困难的，因为该问题的计算复杂度太高，但利用计算机的高速运算能力和穷举算法则很容易获得答案。

求解方法：使用循环结构进行 281631325 次(不含 1 和数本身)重复检查。首先使用一个变量 n 来表示可能的因数，初始值为 2，终值为 281631326，步长为 1，将每次 n 的值作为除数与 281631327 进行相除运算并进行比较。若 n 的值是 281631327 的因数，则进行显示，否则不显示，直到循环次数用完为止。

注意：以上求解方法中的比较次数还可以优化为 281631327 的一半。同时，请读者自行找出另外一个较优化的比较次数，以便提高程序运行效率。

求解算法描述如下：

步骤 1，设定变量 n 的初值为 2；

步骤 2，若变量 $n<281631327$，则执行步骤 3，否则，执行步骤 6；

步骤 3，若变量 n 是 281631327 的因数，则进行显示，否则，不显示；

步骤 4，计算变量 n 加 1 的值并重新赋值给变量 n；

步骤 5，转去执行步骤 2；

步骤 6，算法结束。

说明：在计算机中可以使用表达式 $N\%M==0$，表示整数 M 是整数 N 的因数，其中%符号表示取模运算，即获得两数相除后的余数(取值范围为 $0\sim M-1$)；另外，比较两数是否相等使用是符号==，而符号=通常在计算机中用于表示赋值运算。

下面写出求解此问题的 Python 程序。

```
k=1                                  #设置记数变量的初始值
#穷举全部可能的因数并找出因数
for n in range(2,281631327,1):
    if 281631327%n==0:               #判断是否因数
        print("第",k,"个因数:\t",n)   #显示因数
```

```
        k=k+1                              #记数
```

本例中的第 1 行设置计数变量 k 的初始值为 1；第 3 行使用 for 循环来穷举全部可能的因数，其中的 range()函数指定变量 n 的初始值为 2，终值为 281631326，步长为 1；第 4、5 行是循环体，将每次 n 的值作为除数与 281631327 进行比较，若 n 的值是 281631327 的因数则进行显示，否则不显示；第 6 行让变量 k 进行计数。

运行结果如图 1-5 所示。

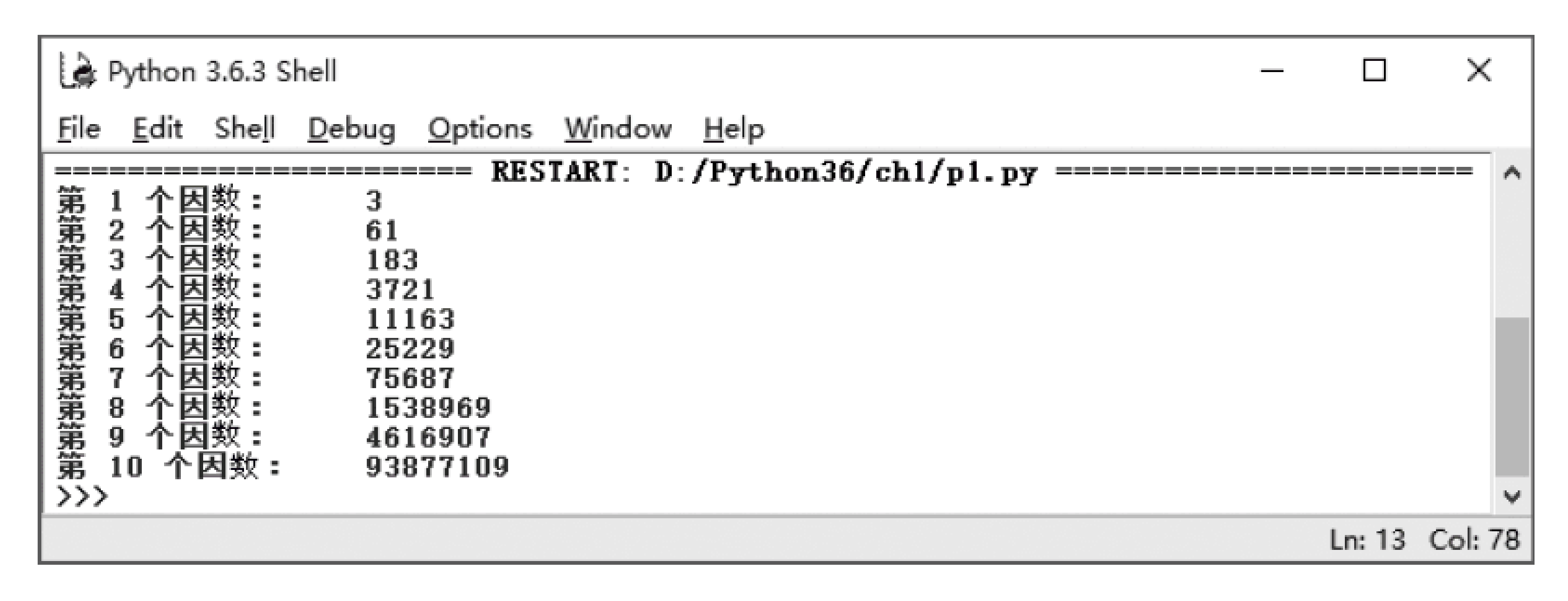


图 1-5　整数 281631327 中的全部因数

注意：由于本例中的循环次数(281631325)太多，系统运行将费时数十秒。

2. 设计系统

在解决任何一个复杂的系统问题时，均需要进行系统设计，即确定系统的逻辑模型和功能要求。众所周知，网络互连系统是非常复杂的综合系统，但其中可以应用计算思维中的分层次设计方法。下面以网络互连系统中的 TCP/IP 协议为例进行说明。

TCP/IP 就是一组专门实现网络互连的通信协议，即 Internet 网络体系结构是以 TCP/IP 协议为标准构建起来的。在 TCP/IP 协议中，使用的系统设计思想就是分层次设计方法，也就是将网络互连系统分成四个层次，分别是应用层、传输层、网际互联层和网络接口层。在这一框架下进一步详细规定每一层的功能，以实现开放系统环境中的互连性、互操作性和可移植性。具体分层如下：

(1) 应用层分为 TELNET、FTP、HTTP、SMTP 等；

(2) 传输层分为 TCP、UDP 等；

(3) 网际互联层分为 IP、ICMP、ARPR、ARP 等；

(4) 网络接口层分为 Ethernet、Token ring、x. 25 等。

从计算思维视角来看，TCP/IP 协议的分层体系充分体现自顶向下、分治、关注点分离等编程思想，最终是将一个复杂系统分解成若干个简单系统，甚至分解过程还可从继续下去，直到系统能够获得设计为止。从制造产品视角来看，相关软、硬件企业可以将 TCP/IP 协议作为计算机业界的“事实标准”，组织生产对应的计算产品，只要严格按标准执行，上下层之间的兼容将自然得到保证，进而推动整个计算机工业的发展。

注意：TCP/IP 协议只是计算机业界的事实标准而不是工业标准。这是由于计算机发展速度极快，业界已不可能有充足的时间去制定标准，即使制定出标准也必将是过时的；二是由于计算机行业规模极大，众多企业和组织要达成一个工业标准将是极为困难的。事实上，近三十年，计算机业界已很少出现传统意义的工业标准了。

3. 人类行为理解

1985 年 ACM 图灵奖获得者理查德·卡普(Richard Karp)认为,自然问题和社会问题本身都蕴含大量基于计算科学的演化规律,这些演化规律伴随着物质、能量、信息等方面的许多变换,所以如果人类能够正确提取这些“变换”并通过可计算的方式描述出来,则显然就可以利用计算机进行处理,这就是基于计算思维解决自然问题和社会问题的基本原理。尽管计算机不能解决物质变换和能量变换方面的问题,但可以借助计算环境进行信息变换,进而预测自然系统和社会系统的演化过程。下面以基于内容的人际关系挖掘为例进行说明。

众所周知,互联网中蕴含着大量的姓名与人际关系方面的数据,但这些大量数据本身即无序也无用,而一个人要试图从中挖掘出既有序又有用的数据是完全不可能的。不过,如果利用文本信息抽取技术,则可以自动地抽取姓名,区分重名,查找或计算人与人之间的关系,进而找出正确的人际关系描述,从而形成互联网世界中的社会关系网,这就是微软亚洲研究院的“人立方搜索”项目正在实现的目标。该系统可以在数十亿的中文网页中自动抽取人名、地名、机构名和短语,并通过算法自动计算出其中存在的关联。人立方搜索的创建理念来自于“六度空间”,例如只要随便输入一个著名人物的姓名,人立方搜索将给出与此人相关的人际联系、性格特征、求学过程、就业情况、学术影响等信息。

综上所述,在社会科学中融入计算思维,是自然科学与社会科学的完美结合,这样可以提高社会运作效率,进而推动社会的发展与进步,同时应验“一切皆可计算”的理念。

1.2.2 计算思维特征

仅就问题求解过程而言,计算思维与数学思维、实验思维、工程思维等均有相似之处。从思维科学的角度看,数学思维是以理性思维为核心的,包含直觉思维、想象力和潜意识思维的问题求解模式,能够孕育理性主义思想;计算思维是以过程性思维为核心的问题求解模式,由一系列规定好的有限步骤组成,其中包括发现问题、确定问题和求解问题,并能够清楚地描述求解问题的操作过程、数据环境、问题限制和作用范围。

1. 计算思维是问题求解思维

将问题求解的过程用“机械化”和“程序化”方式进行表示。问题求解过程分为 5 个阶段:提出问题、分析问题、建立联系、行为选择和反思检验,其中的每个阶段都具有计算思维属性。所以,在面对计算问题时,应该依据已有知识,提出各种问题求解方案,并选择最优算法进行编程,最终由机器运行程序来实现问题求解。

2. 计算思维是确定性的、形式化的科学思维

确定性是计算机算法和程序实现的自然要求,其思维一定会使用确定性符号系统来描述问题并实现求解。确定性表示机器实现时的每个操作都必须是确切定义的,没有任何歧义,这首先由计算机语言本身的完全确定性和可靠性得到保证,可是在自然语言中,经常出现因文化习惯、场景、语调、词语分隔等导致的歧义,例如“读好书”可以理解为只是读一类好书的个体要求,也可理解为好好地读书的整体要求;形式化要求思维过程严格遵循分析逻辑的规律,逐步进行推理,最终获得正确答案,所以计算思维体现的正是严谨的、形式化的逻辑思维,其具体实现需要进行精确的算法描述和严格的符号表示。

3. 计算思维是人机共存思维

计算思维成果最终必须由计算机实现。算法一般分为 3 种形式：生活算法、数学算法和计算机算法。生活算法是指完成某一项工作的方法与步骤，例如大学生的学习计划；数学算法是指对一类计算问题的机械的、统一的求解方法，例如多项式的因式分解过程；计算机算法是指问题求解的精确描述，它具有明显的自动化特征，并且具有计算精度高和操作时序固定的特点，这是与计算机系统本身紧密相关的，所以用计算机实现问题求解时，需要充分利用计算机的高速度、大容量存储、故障率低等优势，尽量发挥计算机与计算思维的威力，例如数学中的许多不定方程就可以使用穷举算法进行求解。读者可以从中体验人与机器不同的计算处理机制，进而加深对机器计算能力和约束范围(与计算机指令系统和编程语言相关)的理解，这将有助于读者在将来的学习、工作和生活中更好地融入信息社会。

4. 计算思维的本质是高度抽象和机器自动实现

计算思维的抽象体现在完全使用形式化的符号系统，这是与代数科学完全一致的。例如，程序由标识符、常数、数据类型、变量名、数组名、运算符、表达式、语句、程序段、函数等语法成分构成，并严格遵守语法定义、语义规范和语用限制，所以编程必将导致思维过程的逻辑严密性；机器自动实现体现在算法实现最终是“机械式”的按步骤地自动执行，这是冯·诺依曼体系结构的本质特征，即存储程序原理。

下面将进入到程序和计算思维的计算机语言实现中，即 Python 语言的运行环境。

1.3 Python 简介

Python 语言属于自由开放的开源软件，任何人都可以自由地使用和发布该语言的副本，阅读全部源程序并进行修改，这为 Python 的推广和应用提供基础。截至 2018 年 1 月，Python 语言已成为最常用的七大编程语言之一，具体排名如表 1-1 所示。

表 1-1 常用编程语言

2018 年 1 月排名	程序设计语言	使用率百分比(%)
1	Java	14.215
2	C	11.037
3	C++	5.603
4	Python	4.678
5	C#	3.754
6	JavaScript	3.465
7	Visual Basic .NET	3.261

资料来源：https://www.tiobe.com/tiobe-index/

1.3.1 Python 优点

Python 应用范围的广泛扩充是基于其优良品质，下面说明主要的优点。

1. 内存回收机制

Java、C++、C#、Python 等现代语言均采用垃圾(内存无用单元)收集机制,而不再是要求用户自行管理内存。不过,Python 使用的是"以引用计数机制为主,以清除与收集两种机制为辅"的管理策略,这样能够充分提高内存效率。由于 Python 语言的数据范围和精度只与内存空间相关,所以保证较大的可用空间将扩展机器的计算能力,例如整数的取值范围就与内存的可用空间相关。

2. 简单

Python 是体现极简主义思想的计算机语言,例如在符号体系、程序描述、语言翻译、操作模式等方面呈现各种简洁思想。阅读一个 Python 程序就好像是在阅读一篇英文文章,通俗易懂。其好处就是,程序员在编程时只需要专注于求解问题本身,而不是化过多时间和精力在语言的语法细节和机器实现方面。

3. 易学

设计 Python 的主要目标就是适合教学的通用语言,其符号体系与英语的描述方式完全一致,具有极佳的可阅读性。一方面,初学者可以如写英文一样地编写程序;另一方面,还可以非常容易地阅读别人编写的程序。

【例 1-3】 易学的 Python 编程。

源程序如下:

```
for line in open("f1_01.txt"):
    for word in line.split():
        if word.endswith("er"):print(word)
```

本例中的第 1 行表示逐次循环读取数据文件中的每一行,其中需要调用 open()函数打开一个文本文件;第 2、3 行是循环体,在处理当前行时判断该行中有无子串 er,若有则显示单词(例如 teacher),否则没有显示。其中,split()函数能够从字符串中取出单词,endswith()函数可以测试单词中有无子串 er。

在运行程序前,要使用 Windows 中的记事本工具建立 f1_01.txt 文件,如图 1-6 所示。

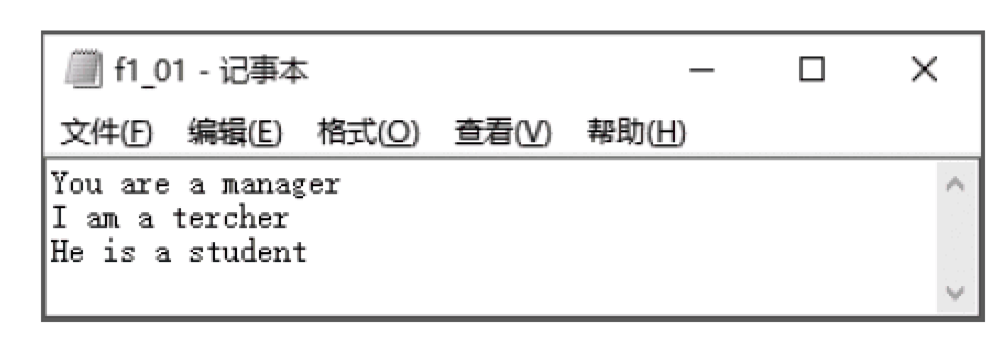


图 1-6 用记事本工具编辑文件 f1_01.txt

运行结果:

```
teacher
```

注:本书中的全部数据文件,均以统一方式指定其对应的文件名称:由字母 f 开头,后面的第 1 个数字表示章的序号,第 2 个数字表示数据文件的序号(两位数字),且二者使用下画线连接,扩展名必须是.txt,例如 f1_01.txt。

4. 高层语言

Python 属于现代语言,因而编程时不需要考虑如何管理程序在内存中运行、CPU 如何

调用程序、程序如何自动执行等底层操作细节，这一点表现出来的效果比 C 语言更高级。让程序员尽量从机器实现的复杂细节中解放出来，才能全力以赴地从事软件开发。

5. 可移植性

Python 的开源本质使其程序很容易移植到许多平台中运行，即一个 Python 可以经过较少改动或不改动就能够在其他平台上运行。这些平台包括 Windows、Linux、OS/2、Macintosh、Windows CE、Solaris、Amiga、Android、AROS、AS/400、OS/390、z/OS、Palm OS、QNX、VMS、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、PocketPC、Symbian 等，如此广泛的平台无关性，使得 Python 成为一个真正全兼容的计算机语言。

6. 解释性

Python 程序不需要翻译成二进制形式的目标代码，就可以直接从源代码中开始运行程序。它的处理过程是，Python 解释器把源程序转换成称为字节码的中间代码，然后再把它翻译成计算机使用的机器语言并运行。其好处是使 Python 程序更加简单易学，也更加易于移植。

7. 面向对象

Python 同时支持面向过程和面向对象两种编程方式。在面向过程编程方面，初学者可以学习各种结构化程序设计方法，尤其是计算思维；在面向对象编程方面，程序员则可以尽量调用各种标准库、扩展库和模块中的方法和属性，进而极大地提高编程效率和增强计算机的应用范围。

8. 扩展性

如果一段程序需要高速运行或者算法需要保密，则可以使用 C 或 C++ 来编程，然后在 Python 程序中加以引用。另外，Python 标准库非常丰富，从而扩展其处理能力，例如正则表达式、文档生成、单元测试、线程、数据库、游戏设计、网页浏览、CGI、FTP、电子邮件、XML、HTML、WAV、密码系统、图形用户界面、Tk 工具、wxPython、Twisted、Python 图像库等，均可以使 Python 增强处理能力。若再加上很容易引用的第三方模块，则使 Python 程序几乎无所不能。

9. 嵌入性

与扩展性对应的是，Python 程序也可以嵌入到 C 或 C++ 程序中实现交叉调用，从而为程序提供脚本(Script)描述功能。

10. 代码规范

Python 采用强制缩进(最好 4 格，本书全文使用 4 格缩进)的方式使得代码具有较好的可读性，从而可以清楚区分程序中的层次划分情况。同时，能够简化代码的书写量。

1.3.2 Python 缺点

事物总是具有两面性的 ，Python 的某些优点必然将导致不足，下面介绍 3 个方面。

1. 运行速度较慢

Python 语言使用的翻译方式是解释方式，固然这样能够方便初学者的学习和编程能力的形成，但必将导致运行速度较慢，这是 Python 的先天不足之一。

2. 单行语句和命令行输出问题

编程时,经常会出现不能将程序写成一行的情况,例如:

```
import sys : for i in sys.path : print(i)
```

但在 Java 和 C 这样的语言中均没有这一约束,而是可以极方便地通过众多分隔符表示一个程序,而 Python 程序使用较新的编程规范。

在 Python 中,以上程序段的规范编码形式如下:

```
import sys
for i in sys.path:
    print(i)
```

3. 语法独特

在将来某时或许这一点不应该被称为缺点,但是目前使用缩进来区分不同程序段的层次,的确会给很多程序员带来困惑,不论是初学者还是高手都需要较长的时间才能适应,这再次说明编程习惯不是非常容易改变的。当然,任何新生事物都有一个逐步被接受的过程。

1.3.3 Python 主要应用

Python 语言的应用范围非常广泛,可称为无处不在,例如操作系统管理、工程与科学计算、网络编程、图形用户界面设计、游戏开发、多线程与并行、文本处理、数据库访问、多媒体应用、搜索引擎等,下面对上述的前四点进行说明。

1. 操作系统管理

Python 提供功能齐备的应用程序编程接口(Application Programming Interface, API),从而让程序员能够非常方便进行系统的组织、管理、调度和日常维护,尤其是多线程管理与并行控制对 Windows 系统提供全面支持。

2. 网络编程

Python 提供丰富的模块支持 Sockets 编程方式,进而使程序员能够方便快捷地编写出各种基于分布式计算原理的应用程序。例如基于开源代码的 Web 应用服务器 Zope、用于 EEG/EMG 的分析软件 Mne、磁力链接搜索引擎 Btgoogle 等。实际上,很多大规模软件开发系统中均在使用 Python 的网络编程方案。另外,Python 还具有 Web 编程能力,支持最新的 XML 技术。

3. 图形用户界面设计

Python 提供 PIL、Tkinter 等图形库,例如支持页面布局、界面设计、图像显示、图形绘制、海龟绘图、控件安排、事件处理等,从而使程序员能够方便地设计图形用户界面。

4. 工程与科学计算

Python 提供大量与许多标准数学库的接口,从而让程序员能够非常方便地引用其中的各种工程与科学计算,不再需要自行编程实现,这也是面向对象技术的体现。例如,SciPy 库属于科学计算,主要致力于科学计算中最常见的问题,像插值、积分、优化、图像处理、矩阵变换、特殊函数等。图 1-7 中所示的就是实现工程与科学计算的 SciPy 库。



图 1-7　实现工程与科学计算的 SciPy 库

1.4　Python 运行环境

在介绍 Python 语言的概况后，下面介绍 Python 的运行环境和操作模式。

1.4.1　Python 下载与安装

1. 下载 Python 安装程序

在安装 Python 前，首先应该访问 Python 官方网站并获取所需的 Python 安装程序。由于本书以操作系统 Windows 10 64 位专业版为例进行介绍，所以选择 Windows 10 平台的 64 位 Python 3.6.3 版本。

具体下载过程说明如下：

(1) 进入 Python 官网 http://www.Python.org/download/，如图 1-8 所示。

基于 Python 语言独立于平台的特点，在 Python 官网中下载能够在 Windows 10 平台下运行的安装程序，程序员应该根据其所用的操作系统位数(32 位还是 64 位)做出正确选择，否则将无法安装。

(2) 本书中选择的是 Windows 10 平台的 64 位 Python 3. 6. 3 版本，如图 1-9 所示。等待下载完成后，就可以进行安装。

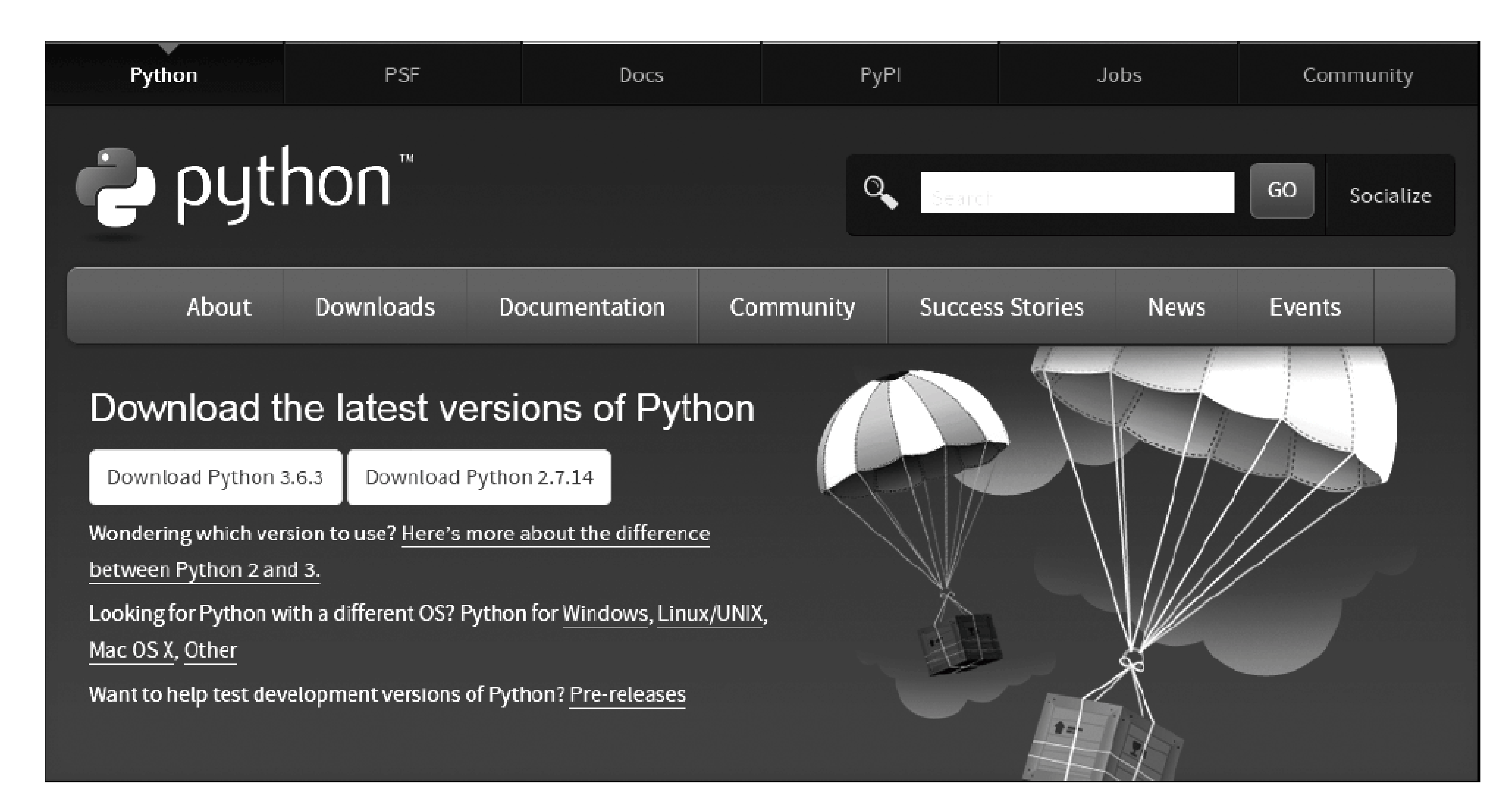


图 1-8　Python 官网主页

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		e9180c69ed9a878a4a8a3ab221e32fa9	22673115	SIG
XZ compressed source tarball	Source release		b9c2c36c33fb89bda1fefd37ad5af9be	16974296	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	ce31f17c952c657244a5cd0cccae34ad	27696231	SIG
Windows help file	Windows		a82270d7193f9fb8554687e7ca342df1	8020197	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64, not Itanium processors	b1daa2a41589d7504117991104b96fe5	7145844	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64, not Itanium processors	89044fb577636803bf49f36371dca09c	31619840	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64, not Itanium processors	b6d61642327f25a5ebd1a7f11a6d3707	1312480	SIG
Windows x86 embeddable zip file	Windows		cf1c75ad7ccf9dec57ba7269198fd56b	6388018	SIG
Windows x86 executable installer	Windows		3811c6d3203358e0c0c6b6677ae980d3	30584520	SIG
Windows x86 web-based installer	Windows		39c2879cecf252d4c935e4f8c3087aa2	1287056	SIG

图 1-9　下载主页(部分截图)

2. 安装 Python 系统

下面说明 Python 系统的安装过程。

(1) 确认安装路径。Python 默认安装路径为 C:\Users\chen\AppData\Local\Programs\Python\Python36-64,当然安装过程中可以改变其默认路径,但本书选择安装路径为 D:\Python36。在运行安装程序过程中,首先就是要确定安装路径。所以,单击复选框 Add Python 3.6 to PATH,选择安装方式为 Customize installation,如图 1-10 所示。

(2) 定制所需安装的组件。默认安装的组件包括 Python 解释器、标准库、说明文档等。可以通过单击相应组件项来定制自己需要安装的组件,而不是沿用 Python 系统的默认设置,如图 1-11 所示。

由于目前软件安装都非常简单,所以程序员只要根据向导一步一步地进行操作即可。其中,安装过程会出现选择安装路径的对话框,如图 1-12 所示。

单击 Install 按钮后,系统将自动进行安装,最终会出现完成安装的对话框。



图 1-10　选择自定义安装与设置安装路径

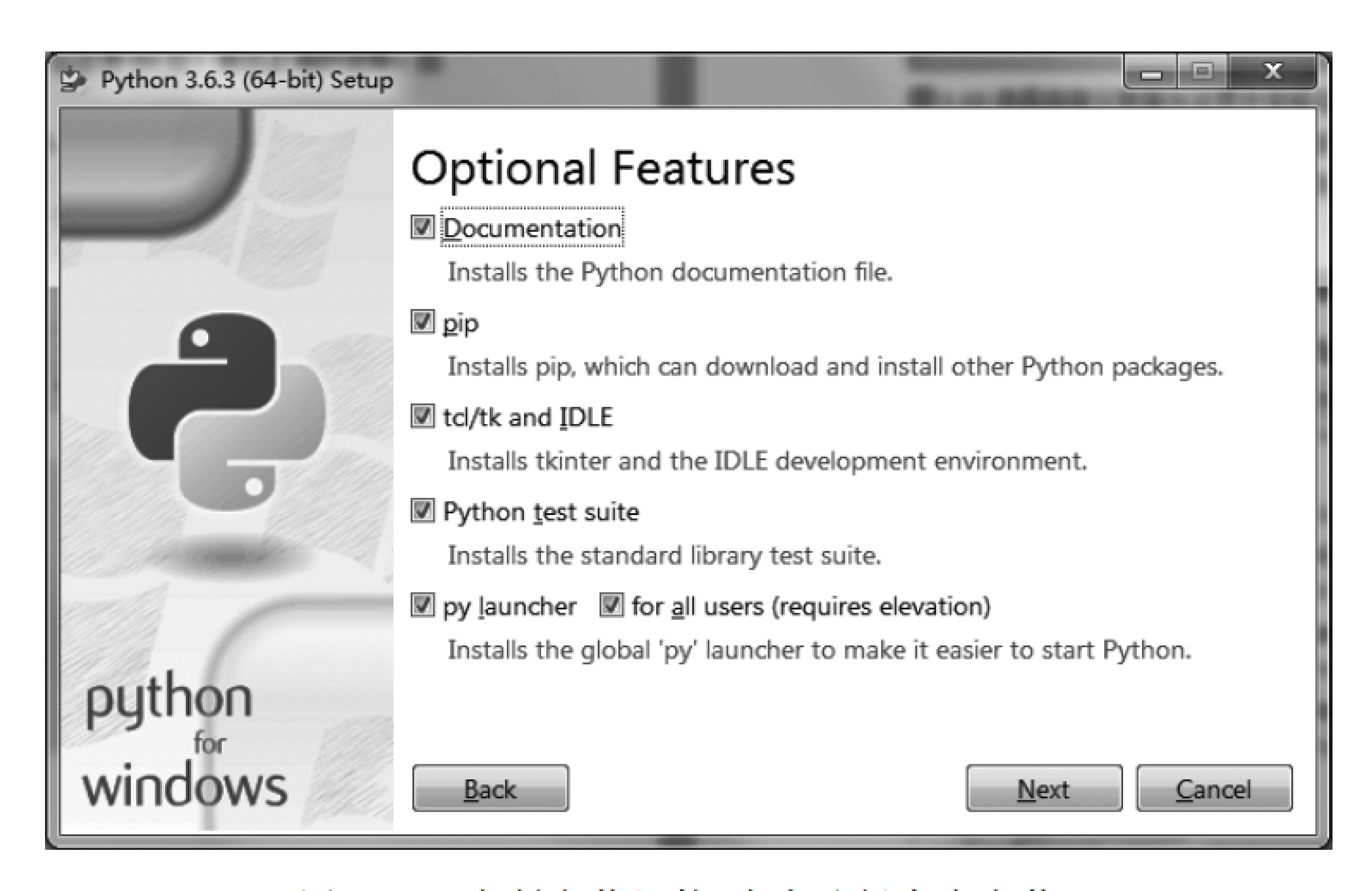


图 1-11　定制安装组件（本书选择完全安装）

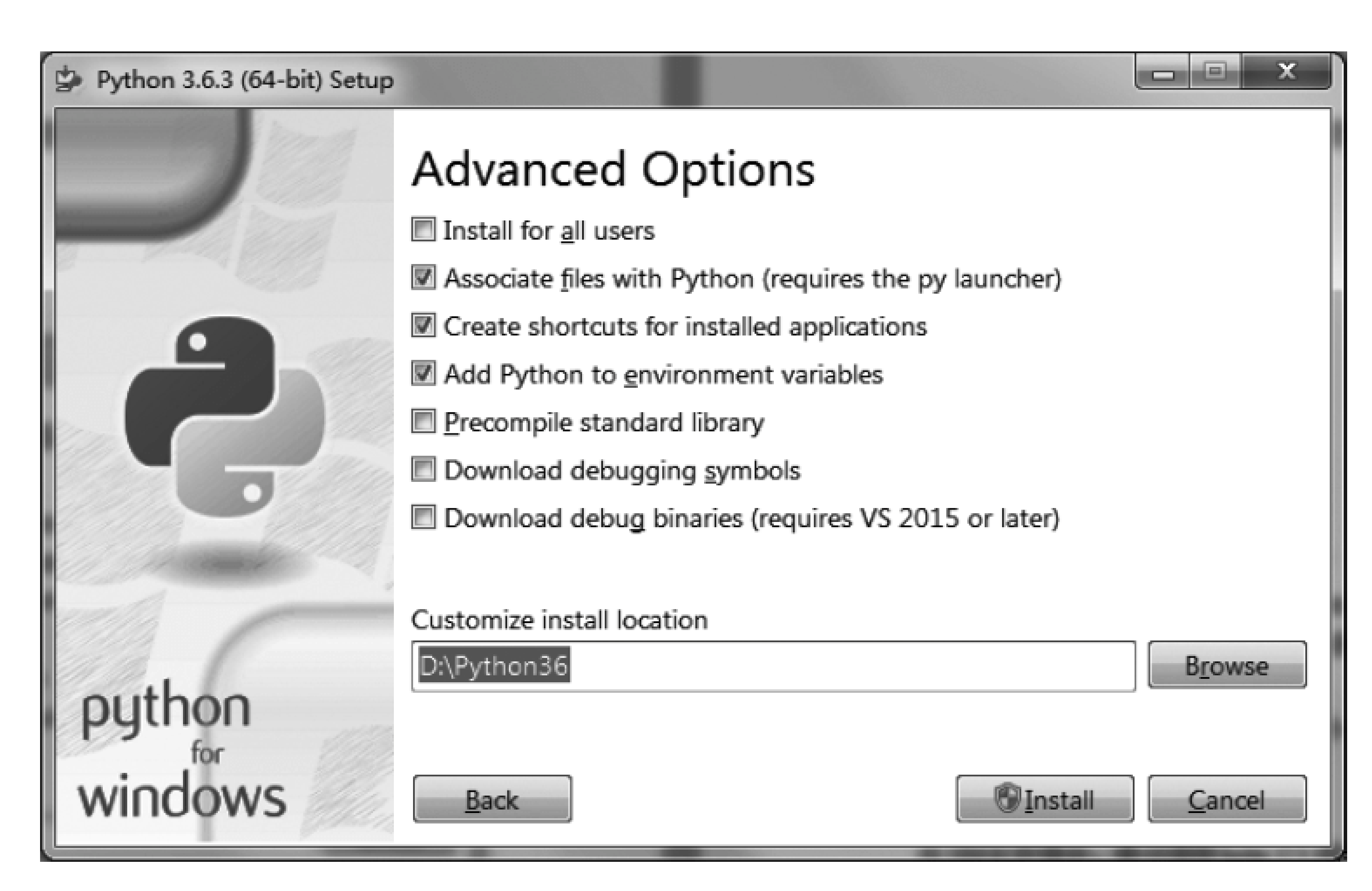


图 1-12　设置安装路径

1.4.2 Python 帮助信息

Python 安装完成后，就可以在“开始”菜单中找到 Python 菜单项，如图 1-13 所示。

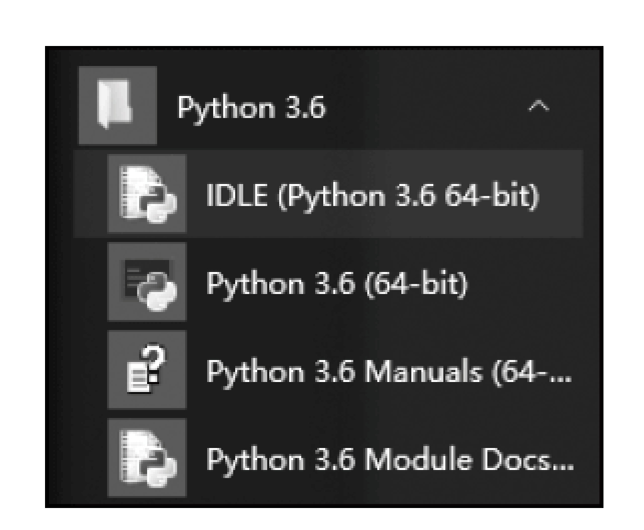


图 1-13　开始菜单中的 Python 菜单项

在图 1-13 中，Manuals 和 Modules Docs 为 Python 的文档和标准手册，是可供随时查阅的文档。从“开始”菜单选择 Python 安装文件夹下的 Python Manuals，就可以打开帮助信息窗口。获取帮助信息是用户自学习 Python 的一种手段，且具有权威性、原创性和真实性。下面介绍 4 种获取帮助信息的方法。

1. 交互式帮助系统

在 IDLE 交互环境中，通过调用 help()函数就可以直接进入 Python 的交互式帮助系统，输入如下命令：

```
>>>help()
```

这里的 3 个大于号(>>>)就是 IDLE 交互环境的提示符。当输入没有参数的 help()函数后，系统将呈现交互式帮助系统，如图 1-14 所示。

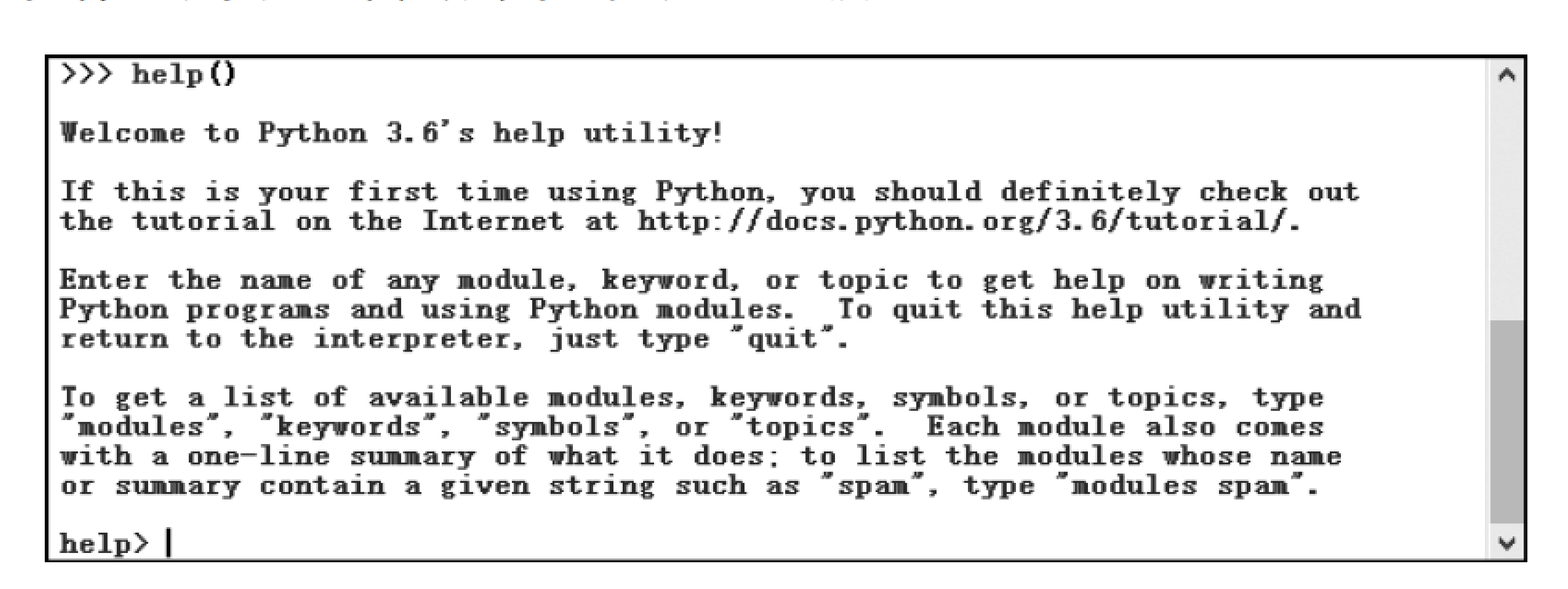


图 1-14　交互式帮助系统

若输入含有参数的 help()函数，例如：

```
help (object)
```

则系统将呈现关于 object 对象的帮助，如图 1-15 所示。

```
>>> help(object)
Help on class object in module builtins:

class object
 |  The most base type
```

图 1-15　object 对象的帮助信息

2. 由内置函数获取帮助信息

要查看 Python 的全部内置函数,则可以在 IDLE 交互环境中,输入如下命令:

```
>>>dir(__builtins__)
```

这里的符号__builtins__前后是两个下画线符号。其后系统将呈现 Python 的全部内置函数,如图 1-16 所示。

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

图 1-16 Python 的全部内置函数

若要继续查看关于 int 类型的详细信息,则可以输入如下命令:

```
help(int)
```

系统将呈现 int 的帮助信息,如图 1-17 所示。

```
>>> help(int)
Help on class int in module builtins:

class int(object)
 |  int(x=0) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
 |  4
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
```

图 1-17 int 的帮助信息(部分截图)

3. 获取 Python 文档

Python 文档是对指定版本的语言规范和标准模块的详细描述,显然这是代表设计者的

权威和标准,可作为读者学习 Python 编程的重要工具。

具体操作是,选中“开始”|Python 3.6|Python 3.6 Manuals(64-bit)菜单选项,就可以打开 Python 文档,如图 1-18 所示。

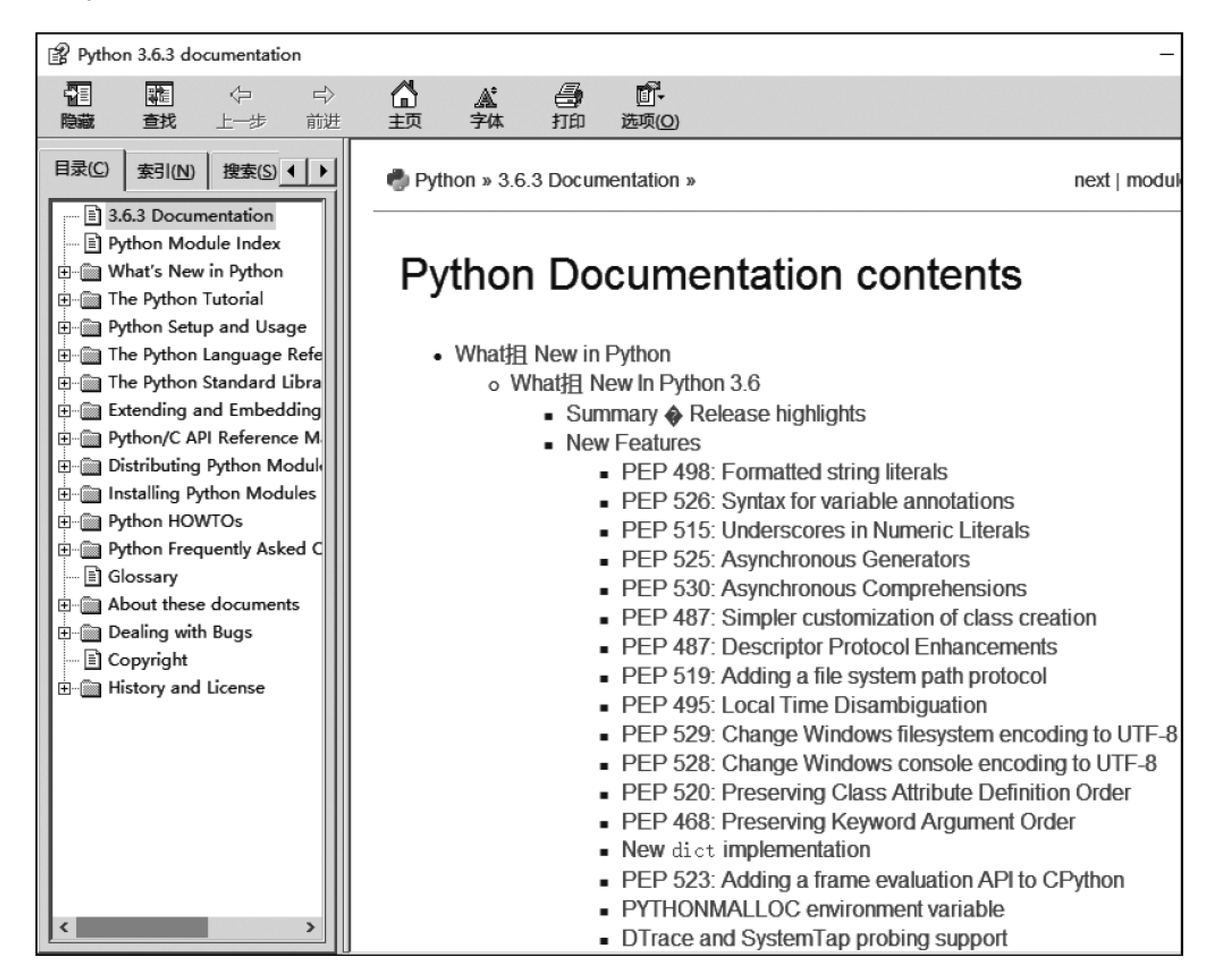


图 1-18 Python 文档

4. 获取 Python 文档的文件

Python 文档以文件名 Python 3.6.3 Documentation.CHM 存放在 D:\Python36\Doc 文件夹中。该文件是一个已编译的 HTML 帮助文件,读者可以直接用 IE 或 Edege 浏览器进行阅读,尤其是 Python 教程(英文名称是 The Python Tutorial)部分,如图 1-19 所示。

1.4.3 Python 文件夹结构

以 Python 3.6.3 版本为例,安装到系统后获得的文件夹结构如图 1-20 所示。

Python 文件夹结构,如表 1-2 所示。

由于 Python 语言的开源性质,所以在 Python 的系统文件中有许多源程序文件,读者可以直接引用,其中的海龟绘图程序可以参看第 11 章。

1.4.4 Python 运行模式

Python 提供 3 类共 5 种运行模式,以支持用户的各种操作需求,下面分别进行描述。

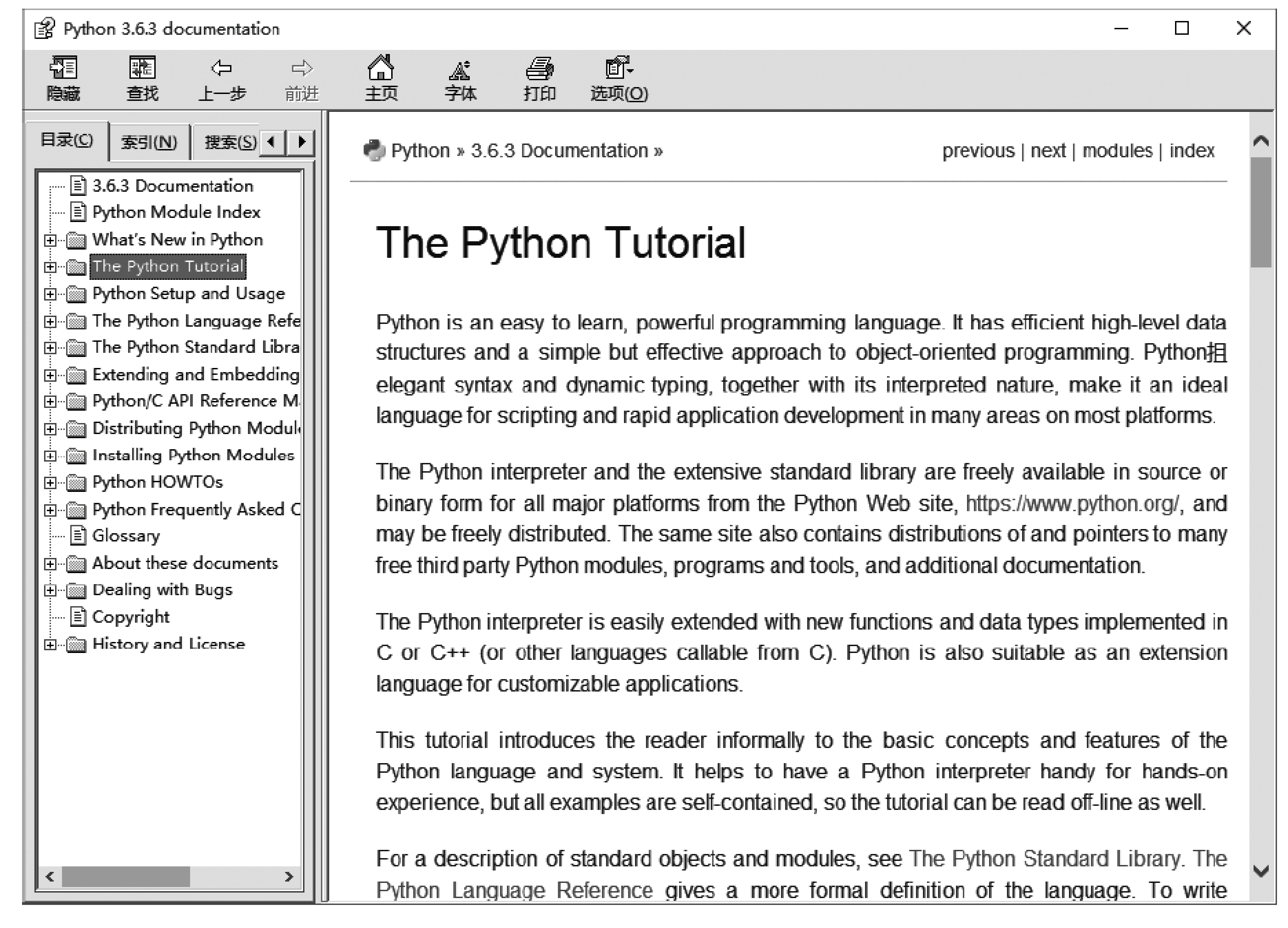


图 1-19 Python 教程(The Python Tutorial)

此电脑 › 本地磁盘 (D:) › Python36 ›

快速访问
OneDrive
此电脑
WPS云文档
视频
图片
文档
下载

名称	修改日期	类型
DLLs	2017/12/3 13:48	文件夹
Doc	2017/12/3 13:47	文件夹
include	2017/12/3 13:47	文件夹
Lib	2017/12/3 13:47	文件夹
libs	2017/12/3 13:47	文件夹
Scripts	2017/12/3 13:48	文件夹
tcl	2017/12/3 13:47	文件夹
Tools	2017/12/3 13:47	文件夹
python.pdb	2017/10/3 18:15	PDB 文件
python_d.pdb	2017/10/3 18:17	PDB 文件

7 个项目

图 1-20 Python 的文件夹结构(截图未完,其后全部为普通文件)

表 1-2 Python 文件夹结构

文件夹	说 明
DLLs	用于 Python 扩展模块、共用图标、应用程序扩展等
Doc	编译的 HTML 帮助文件
include	可由 Python 程序调用的 C/C++ 程序文件
Lib	各种系统库文件和示例程序
Libs	各种扩展的系统库文件
Scripts	各种脚本应用程序
Tcl	各种系统库文件
Tools	各种工具包和演示程序

1. IDLE 集成开发环境

IDLE(Integrated Development and Learning Environment,集成开发学习环境)是Python 语言专用的、具有图形界面效果的集成开发环境,以便实现 Python 程序的编写、设计、输入、修改、调试等。选中“开始”|Python 3.6| IDLE(Python 3.6 64-bit)菜单选项,就可以进入 IDLE 窗口,如图 1-21 所示。

```
Python 3.6.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
                                                              Ln: 3  Col: 4
```

图 1-21　IDLE 窗口(Shell)

IDLE 集成开发环境具有人机交互能力。支持全部的文本编辑操作,例如输入语句并构成程序文件。另外,IDLE 窗口还支持 Windows 系统中的复制、剪切和粘贴操作,这极大地方便了程序编辑操作。

在 IDLE 集成开发环境有两种运行模式:交互环境运行模式和编程环境运行模式。下面分别进行介绍。

(1) 交互环境运行模式。在提示符＞＞＞ 后,输入常量、有值的变量、表达式或语句,这些语法成分都将由 Python 系统自动执行。若语法检查正确,则会显示对应的运算结果,就像使用计算器一样简单,如图 1-22 所示。

```
Python 3.6.3 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> side1=5
>>> side2=6
>>> area=side1*side2
>>> print("面积: ",area)
面积:  30
>>> |
                                                              Ln: 8  Col: 4
```

图 1-22　IDLE 交互环境运行模式

Python 具有智能识别功能,可用不同的颜色对代码进行标识,其中字符串常量是以双(单)引号界定的绿色的文字,紫色的文字表示 Python 中的函数调用,黑色的文字表示 Python 中的其他语法成分,蓝色的文字是系统运行程序后的输出信息。

注意:

① Python 要求在使用变量前必须先定义变量并指定内容,若语句中引用没有定义的变量,则系统会因没有找到相应的变量定义而提示名字出现错误的信息。

② 既然是人机交互模式,则系统一次只能执行一条语句(即当前语句),也就没有上下文相关的要求。相应的输入语句也不会被保存,所以如果需要编写一个较大型程序,则只有利用文本编辑器来构成程序文件,才可以保存到磁盘中。

(2) 编程环境运行模式。如果将程序组织成文件,则应该在 IDLE 交互环境中选中 File |New File 菜单选项,新建一个编辑窗口来完成程序的输入与修改,如图 1-23 所示。



图 1-23　IDLE 编辑窗口

在 Python 程序中,语句缩进表示程序中的层次关系,不同缩进(本书中的每个层次均以左缩进 4 格为准)表示语句块中的不同层次关系,而层次关系本身就是操作,也是和语句一样是程序的重要组成部分之一。

在程序输入完成后,要用选中 File|Save 菜单选项保存文件,且文件类型为.py。

编辑窗口中的 Edit(编辑)菜单提供许多常规的编辑操作命令。另外,编辑窗口内含智能识别功能,不同的数据类型、保留字、内置函数、语句、注释等会以不同的颜色加以区分显示,以帮助编程者及时发现程序输入过程中的语法错误。例如,字符串常量是以双(单)引号界定的,输入引号后,后面的内容自动变为绿色,表示字符串常量。

注:程序中的红色文字表示程序中的注释,黄色文字表示 Python 中的保留字,紫色文字表示 Python 中的函数调用,黑色文字表示 Python 中的其他语法成分。

注意:*颜色体系是可以在 Settings 对话框中自行设置的。*

在 Windows 窗口环境下,只要双击 Python 程序文件就可以运行 Python 程序。作为程序运行方式,这样会遇到一些问题,例如一个程序没有运行结果(甚至含有部分关于出错的提示信息)会一闪而过,不能像在命令行窗口中运行程序那样,通过将中间结果呈现在屏幕上以实现人机交互。如果想得到程序的运行结果,一种解决方法就是在程序文件的最后添加一条 input()语句(等待输入数据)表示暂停。

如果程序员要求在 IDLE 交互环境下,不仅有编辑程序的环境同时还具有调试程序的功能,则可以选中 Run|Run Modules 菜单选项,就可执行正在编辑的程序,也可以直接按 F5 键,其间系统将要求保存文件。运行结果和异常等都出现在 IDLE 交互环境中,如图 1-24 所示。

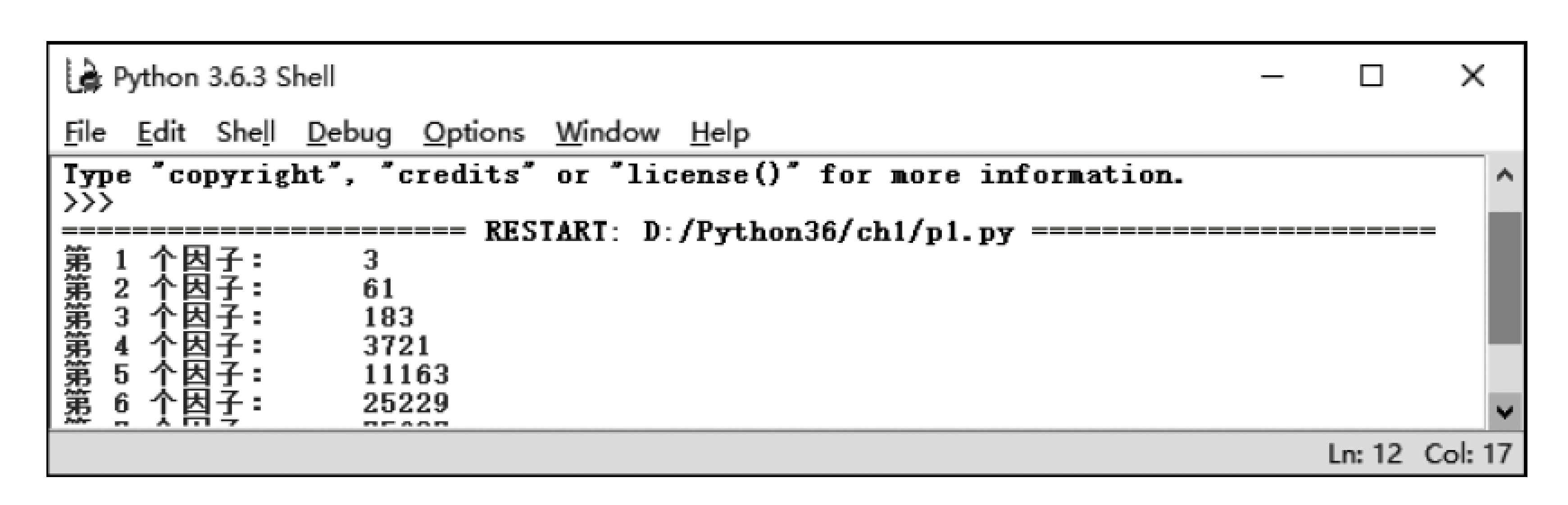


图 1-24　IDLE 集成开发环境(含程序运行结果)

注:为保持全书的描述准确和一致,以便读者能够获取真实的窗口显示,本书对 IDLE 集成开发窗口进行统一设置。操作过程是,在图 1-23 中选中 Options|Configure IDLE 菜单

选项，系统将显示如图 1-25 所示的 Setting 对话框。

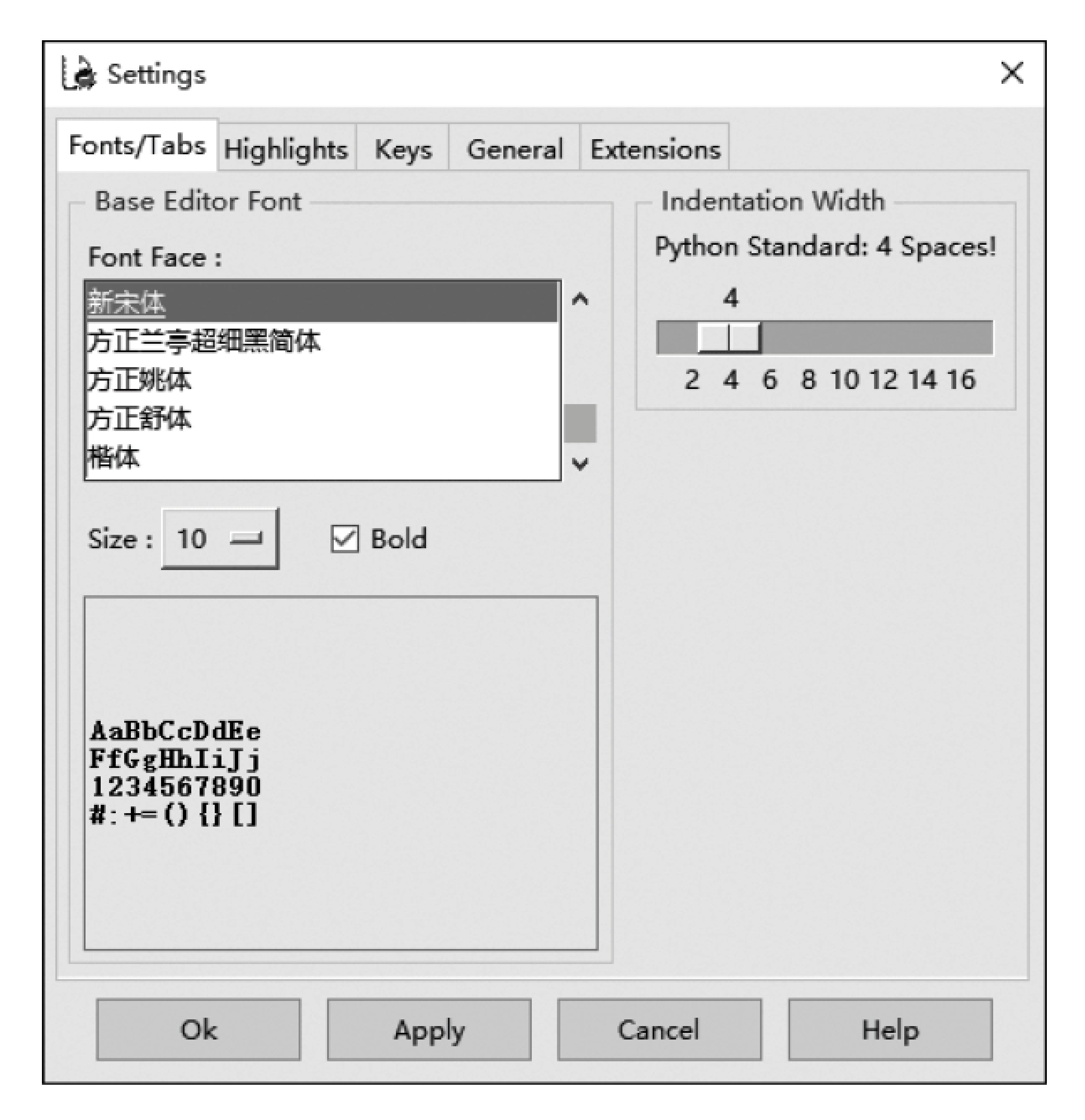


图 1-25　Settings 对话框

在 Settings 对话框中选中 Fonts/Tabs 选项卡，分别设置 Font Face(字体)为“新宋体”，Size(字号)为 10，选中 Bold 复选框，并沿用 Indentation Width(左缩进空格数)为 4。

2. 命令行窗口运行模式

Python 系统提供两种交互运行环境，程序员可根据具体情况进行合理选择。一种是在命令行窗口输入 Python 并回车后的交互运行环境，另一种则是在 IDLE 窗口进入并默认其打开的交互运行环境。

选中“开始”|“所有程序”|Python 3.6|Python 3.6(64-bit)菜单选项，进入命令行窗口运行模式，如图 1-26 所示。

```
Python 3.6 (64-bit)
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 b
64)] on win32
Type "help", "copyright", "credits" or "license" for more information
>>> r=5
>>> area=3.14*5*5
>>> print("面积: ",area)
```

图 1-26　命令行窗口(Python 3.6(64-bit))

在该窗口中，可以直接输入 Python 命令，系统将自动响应，与 IDLE 窗口相同。本例中是提供半径值后，计算圆面积并显示。

注：为保持全书的描述准确和一致，以便读者能够获取真实的窗口显示，本书对命令行窗口进行统一设置。操作过程是，在图 1-26 中，单击标题栏左侧的图标，选择快捷菜单的“属性”选项，系统将显示如图 1-27 所示的 Python3.6(64-bit)属性对话框。

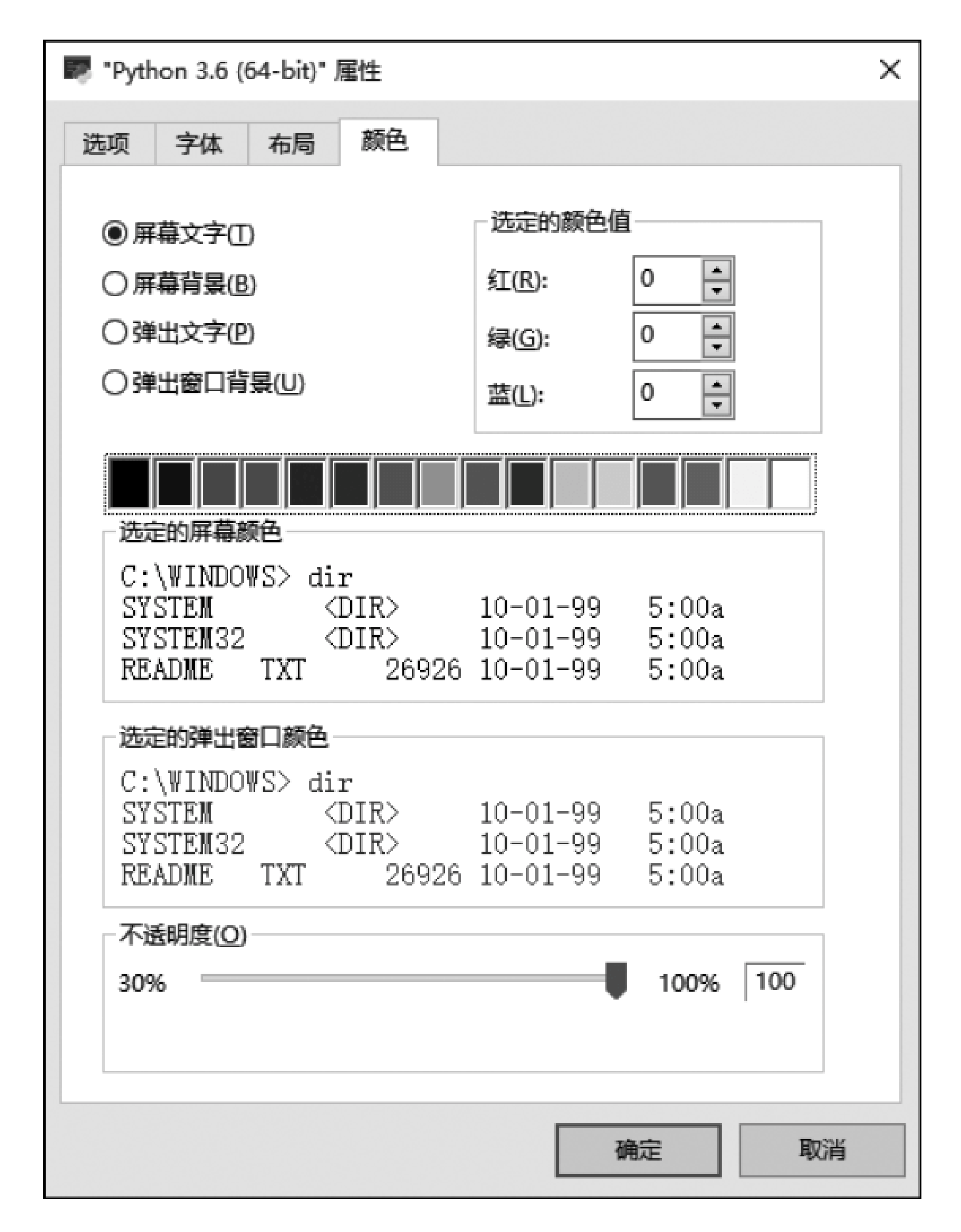


图 1-27 “Python3.6(64-bit)属性”对话框

在“颜色”选择卡中,设置屏幕文字为白色,屏幕背景为黑色。

3. 命令指示符窗口运行模式

(1) 命令指示符窗口运行模式。右击“开始”菜单,从弹出的快捷菜单中选中“运行”选项,系统将弹出“运行”对话框,如图 1-28 所示。

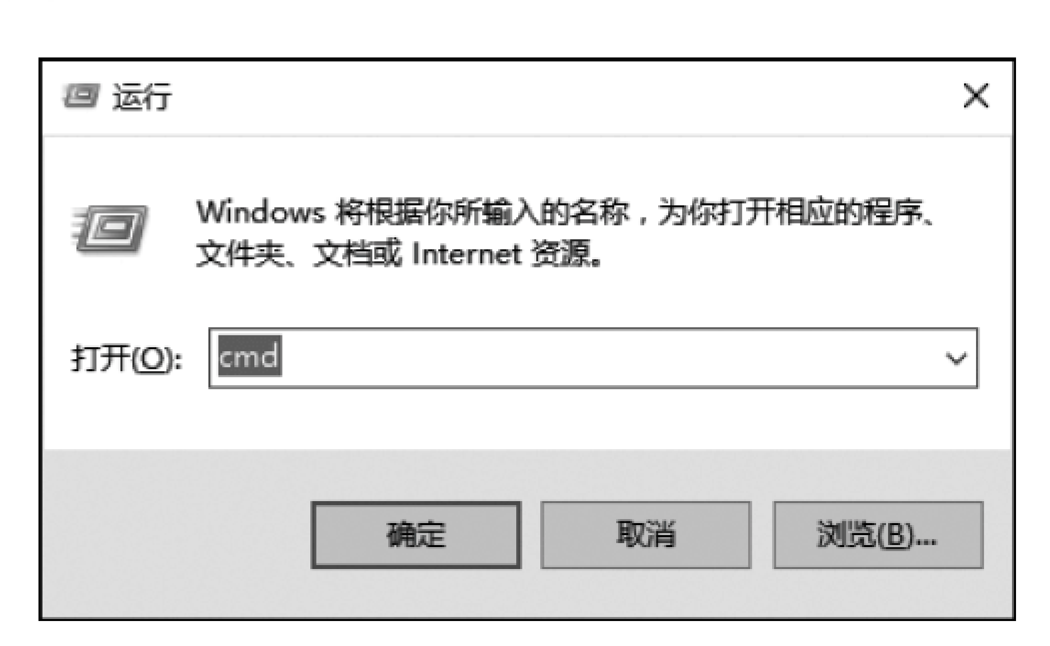


图 1-28 运行对话框

输入 cmd 命令,其后系统将显示进入命令指示符窗口,如图 1-29 所示。

在图 1-29 中,需要输入如下 DOS 命令(提示符是“盘符:>”符号)才能使用 Python。

```
>d:
>cd Python36
```

若要执行程序,可以输入命令“Python p1.py”,其中 p1.py 是程序文件名。

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>D:

D:\>CD Python36

D:\Python36>Python ch1\p1.py
键盘输入数据:
a=1
b=2
c=-3
显示计算结果:
方程的第1个根： -3.0
方程的第2个根： 1.0

D:\Python36>
```

图 1-29　命令指示符窗口

(2) 内含的 IDLE 集成开发窗口模式。若要进入 IDLE 集成开发窗口模式，可以输入没有参数的 Python 命令，如图 1-30 所示。

```
C:\Windows\system32\cmd.exe - Python
Microsoft Windows [版本 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>d:

D:\>cd \Python36

D:\Python36>Python
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 b
Type "help", "copyright", "credits" or "license" for more information
>>>
```

图 1-30　内含的 IDLE 集成开发窗口

注意：以上操作均要求系统安装的路径是 D:\Python36。

以上介绍 Python 的 5 种运行模式，以便支持用户的各种操作需求。由于本书主要目的是讲解编程方法和计算思维，所以通常使用 IDLE 集成开发窗口和编辑窗口。

习　题　1

一、简答题

1. 什么是计算机程序？什么是程序设计？
2. 从键盘输入圆的半径，然后写出分别求解周长、面积和球体体积的操作步骤。
3. 为什么程序设计要分步骤实现？有哪些程序设计步骤？
4. 程序设计语言可以分为哪几代？每一代各有什么特点？
5. 什么是将源程序翻译成目标程序的编译方式？有何特点？
6. 什么是将源程序翻译并执行的解释方式？Python 为什么要使用解释方式？
7. 什么是计算思维？有何主要内容？

8. 简述 Python 语言的主要优点。

9. 简述 Python 语言的主要缺点。

10. 简述 Python 语言的主要应用。

11. 简述 Python 系统的文件夹结构。

12. Python 有哪些运行模式？各有什么特点？

13. 将正整数 n 分解成 $p \cdot q$,要求 p 和 q 的值是最接近的两数。例如,整数 69741 共 11 组分解结果,即 243×287=189×369=…=3×23247,但最接近的两数是 243×287。要求写出求解问题的操作步骤。

二、编程题

1. 参考例 1-1 中的程序,编程求出 3 个实数的平均值。

2. 从键盘输入角度,编程计算对应的弧度。

三、操作题

1. 在“百度”网站中检索“设计系统”词条,就计算思维中的设计系统思想写一篇 600 字的综述文章。

2. 在“百度”网站中检索“计算社会科学”词条,就计算思维中的人类行为理解思想写一篇 600 字的综述文章。

3. 根据所用计算机的操作系统类型和二进制位数,访问 Python 官方网站并下载正确版本的 Python 安装程序,并安装到 D:\Python36 文件夹中。

4. 如何获取 Python 的帮助系统？通过上机操作访问帮助系统。

5. 使用一种 Python 的交互运行模式,显示信息:“欢迎学习 Python 程序设计。”

第 2 章　算　　法

计算机求解问题的操作方式就是算法，任何计算问题都会被分解成算法中的若干操作步骤来实现。本章首先介绍两种完全不同的程序设计方法；其后介绍算法概念以及一些常用问题的算法，例如交换两个变量内容、取绝对值、阶加、阶乘、求最大公约数、求斐波那契数列、判断素数等；最后介绍一些综合算法方面的运用，例如数值计算、穷举算法、查找算法和排序算法，以及迭代、递推、递归等方面的算法。

本章介绍的所有算法均将体现计算思维，即利用准确的抽象与有效的操作时序，最终实现计算机的自动运行。

2.1　程序设计方法

在介绍算法前，首先介绍两种程序设计方法：结构化程序设计方法和面向对象程序设计方法。它们的关系紧密，算法是程序设计的思维基础，程序设计是算法的机器实现前提。

2.1.1　结构化程序设计方法

结构化（又称为面向过程）程序设计是进行以模块功能和处理过程设计为主的设计方法，其概念由迪克斯特拉（E. W. Dijkstra）于 1965 年首次提出，这是软件工程领域的一个重要里程碑。主要思想如下：

（1）采用自顶向下、逐步求精的功能分解方法；

（2）使用 3 种基本控制结构来构造程序，即任何程序均可以由顺序结构、选择结构和循环结构来构造。

1. 结构化程序设计原则

结构化程序设计原则包括自顶向下、逐步细化、模块化和限制使用 goto 语句。

（1）自顶向下。在编程时，绝对不能直接书写代码，而是应该首先考虑总体结构，然后考虑实现细节；换言之，应首先考虑整体目标，然后考虑局部目标，即不要一开始就过多地追求细节，而应先从最上层的整体目标开始进行设计，然后逐步将计算问题的操作过程具体化。正如写文章时，首先需要确定章节划分一样。

（2）逐步细化。在处理问题时，要将复杂问题先分解成若干个子问题，再将子问题分解成更小的若干个子问题，直到将复杂问题转换成可以处理的简单问题时为止，这就是逐步细化。在程序设计层面，最终细化的程度应该到语句能够表达时为止。

（3）模块化。众所周知，任何复杂问题都是由一系列较简单的问题构成的。模块化就是将要解决的总目标分解为若干个子目标，再将子目标进一步分解为具体的小目标，这些子目标或小目标就称为模块。

（4）限制使用 goto 语句。goto 语句可以直接改变程序的运行过程，进而造成程序的逻辑混乱，这必将与人类的线性思维方式相冲突，所以应该限制使用 goto 语句。取消或限制

使用 goto 语句后，程序则将容易理解、排错和维护。Python 作为现代语言，是不使用 goto 语句的。

2. 基本结构

结构化程序是由顺序结构、选择结构和循环结构 3 种基本结构构成的。

(1) 顺序结构。顺序结构表示程序中的每个操作是按照它们出现的先后顺序执行的，常见的输入、计算、输出的程序就是顺序结构。例如，求解一元二次方程时，首先输入 3 个系数，然后用数学公式计算两个解，最后输出两个解。当然，大多数程序不会如此简单，通常情况是顺序结构作为程序的一部分，与其他结构一起构成一个复杂程序，例如选择结构中的复合语句、循环结构中的循环体等。

(2) 选择结构。顺序结构的程序虽然能够解决输入、计算、输出等问题，但不能做出判断并进行选择，对于要先做判断后进行选择的问题就要使用选择结构。选择结构的执行是依据指定条件选择执行路径，而不是严格按照语句出现的物理顺序。关键在于构造合适的分支条件来控制程序的运行流程，即根据不同的程序流程选择适当的选择语句。

选择结构一般分为单分支选择、双分支选择和多选择分支 3 种形式。

(3) 循环结构。循环结构表示程序要重复执行某些操作，直到某条件为假(或为真)时才会终止循环。循环结构可以减少源程序重复书写的工作量。在循环结构中最主要的是要明确哪些操作需要循环执行，什么情况下执行循环，什么情况下退出循环。

循环结构分为两种形式：当型循环和直到型循环。

① 当型循环表示先进行条件判断，当满足给定的条件后才执行循环体，并且在循环终端处流程自动返回到循环入口处；如果条件不满足，则退出循环体直接到达循环出口处。因为是“当条件满足时执行循环体”，即先判断后循环，所以称为当型循环。

② 直到型循环表示从循环入口处直接执行循环体，在循环终端处判断条件。如果条件不满足，返回入口处继续执行循环体，直到条件为真时再退出循环到达循环出口处，是先执行循环后进行条件判断。因为是“直到条件为真时退出循环”，所以称为直到型循环。

3. 结构化程序特点

通过以上描述，就会发现结构化程序的 4 个特点。

(1) 结构化程序内没有无穷循环。

(2) 每种结构都有且仅有一个入口。

(3) 每种结构都有且仅有一个出口。

(4) 结构化程序内的每一部分都有机会被执行到。也就是说，对每一个结构来说，都应当有一条从入口到出口的路径通过它。

2.1.2 面向对象程序设计方法

1. 引例

要解决结构化程序设计方法的可扩展性差、维护代价高、抽象程度强等缺点，只有使用面向对象的程序设计方法。这种方法是通过增加软件的扩充性和重用性来提高程序员编程能力的，它的最大优点是软件具有重用性，这种新技术更接近人的思维活动。人们在利用面向对象思想编程时，可以很大程度地提高编程效率，减少软件维护的开销。当人们对软件系统的要求发生变化时，并不需要程序员做大量的修改工作。

【例 2-1】 从键盘输入正整数 n，计算$\sqrt{n}$的值。

编写 Python 源程序如下：

```
import math                  #导入 math 模块
n=int(input("n="))           #输入数据
result=math.sqrt(n)          #计算平方根
print("平方根:",result)      #输出结果
```

在本例中，计算平方根是通过调用函数 sqrt()实现的，这样使程序员没有必要去编写计算平方根的具体程序。实际上，Python 将许多函数封装在语言系统内部，用户可以自由地进行调用。

本例中的第 1 行导入数学模块 math，以便后续语句能够调用开平方根函数 sqrt()；第 2 行调用 input()函数实现键盘输入，这里的 int()函数实现字符串数据到整型数据的转换；第 3 行调用数学函数 sqrt()计算平方根；第 4 行显示计算结果并结束程序。运行结果如图 2-1 所示。



```
>>>
====================== RESTART: D:\Python36\ch2\p1.py ======================
n=5
平方根: 2.23606797749979
>>>
```

图 2-1　例 2-1 的运行结果

2. 对象与类

对象与类是面向对象程序设计技术中最重要的概念之一，也是学习面向对象程序设计技术的重点，程序员想要掌握面向对象程序设计的技术，首先就要很好地理解类与对象的概念。

(1) 对象。在现实世界中，对象就是人们认识世界的基本单元，它可以是人，也可以是物，还可以是一件事。整个现实世界就是由各种各样的“对象”构成的，对象既可以很简单，也可以很复杂，只不过复杂对象可以由若干简单对象构成。例如，一个苹果、一辆汽车、一个足球、一个学生、一次游行等都可以看成是一个对象。

对象作为现实世界中的一个实体，主要特性如下：

① 每一个对象必须有一个名称以便区别于其他对象。

② 用状态或属性可以描述对象具有的特征。

③ 对象含有一组操作，每个操作都表示对象的一种行为。

④ 对象中的操作与属性是不可分离的。

(2) 类。在现实世界中，“类”就是对一组具有共同属性和行为的对象所进行的抽象，类和对象之间的关系是抽象和具体的关系。类是对多个对象进行综合抽象的结果，对象又是类的个体实现，或者说一个对象就是类的一个实例。例如，由若干个的苹果可以构成苹果类，而一个苹果只是苹果类的一个对象实例。

【例 2-2】 学生类中的一个对象实例，如表 2-1 所示。

本书第 8 章将专门介绍面向对象编程，这里只进行简单说明。

Python 同时支持结构化程序设计方法和面向对象程序设计方法。

表 2-1　学生类中的一个对象实例

属性	属性值	操作	属性	属性值	操作
年龄	18 年	看电影	职务	班长	去听课
年级	二年级	踢足球	专业	计算机科学与技术	打游戏

2.2　算　法

在介绍两种程序设计方法后，还不能立刻编写程序。编程前还应该考虑程序的组织与结构，这就是算法。在算法设计完成后，才可以编写程序。

2.2.1　求解问题方式

在介绍算法前，首先来看人类求解问题的两种方式：推理方式和算法方式。

1. 推理方式

推理方式是由一个或若干个已知判断推出另一个新判断的问题求解方式。例如，“所有哲学家都是智慧的”，由于“苏格拉底是哲学家”，所以就可以推导出“苏格拉底是智慧的”结论。其中，“所有哲学家都是智慧的”和“苏格拉底是哲学家”是两个已知判断，从这两个判断推出“苏格拉底是智慧的”的新判断。简言之，任何一个推理都应该包含已知判断、新判断和推理过程。这种推理方式在数学领域中得到全面应用，即从已知的定理、引理和公理系统出发，用数学演绎的方法，推导出问题的解答。

2. 算法方式

算法方式是构造一个问题求解的基本操作序列，按一定的顺序经过一步一步地“机械”执行操作的过程，最终得到问题解答。当然，该解答是人类可以验证并接受的，它不需要也不可能用数学推理方式进行证明，这是因为二者具有完全不同的公理体系和价值判断。

2.2.2　算法概念

图灵奖得主 D. E. Knuth 对算法的定义是，一个算法就是一个有穷规则的集合，其中规则规定解决了某一特定类型问题的操作序列。在学习程序设计类课程时，要用计算机语言来实现算法设计，体验问题求解的思维训练。算法的操作时序确保了问题求解过程是按步骤进行的，这种执行规则非常简单且机械，所以在学习过程中要尽量多经历算法化过程，并从中体验计算思维，它有利于培养理性思维和形式逻辑能力。

2.2.3　算法特征

一个完整的算法应该具有如下 5 个特征：有穷性、确定性、输入项、输出项和有效性。

1. 有穷性

有穷性是指算法必须在有限的操作步骤后结束。当然，这里的“有限”并不是数学意义上的，而是现实意义上的，例如一个算法的操作步骤不要超过 3min 或 1×10^8 次相加运算。

2. 确定性

算法的每个操作步骤都必须是清楚定义的，不能有二义或歧义。例如在计算机运算过

程中,可以通过确定运算的优先次序和结合方式来保证算法操作的唯一性。

3. 输入项

一个算法有 0 个或多个输入,以便提供算法操作时所需的初始数据。若算法没有输入,则算法本身具有初始数据,其运算过程没有通用性;若算法有输入,则算法将具有通用性,即具有更大处理能力和范围。

4. 输出项

一个算法必须有一个或多个输出,以便表示算法操作后的结果,所以没有输出的算法是毫无意义的。

5. 有效性

有效性是指算法中的任何操作步骤都是能被计算机实现的,这是与计算机的属性和能力相关的,例如目前利用微型计算机计算 1000000!的精确值是不可能的。

2.3 算法表示

算法的常用表示方法有如下 5 种:

(1) 使用自然语言描述算法;

(2) 使用传统流程图描述算法;

(3) 使用 N-S 图描述算法;

(4) 使用伪代码描述算法;

(5) 使用高级语言描述算法。

下面使用以上的前 4 种方法来描述阶加算法,计算 1+2+3+…+99+100,而本书的后续章节则是使用 Python 语言描述算法。

2.3.1 使用自然语言描述算法

在早期的计算机编程过程中,并没有任何形式化的语言可以利用,人们只能借助类似于自然语言的文字符号来描述算法。

【例 2-3】 使用自然语言描述阶加算法。

求解方法:使用循环结构来表示 100 次相加运算。其中,使用两个变量 sum 和 n,变量 sum 表示阶加变量,初始值为 0,每次加上一个“加数”;变量 n 表示“加数”,初始值为 1,取值范围是 1~101,为阶加变量 sum 准备数据,而 n 的值为 101 时表示退出循环。

算法描述如下:

步骤 1,设定 sum 的初值为 0;

步骤 2,设定 n 的初值为 1;

步骤 3,若 $n \leqslant 100$,则执行步骤 4,否则转出执行步骤 7;

步骤 4,计算 sum 加 n 的值并赋给 sum;

步骤 5,计算 n 加 1 的值并赋给 n;

步骤 6,转去执行步骤 3;

步骤 7,输出 sum 的值;

步骤 8,算法结束。

从以上描述的求解过程中可以发现，用自然语言描述的算法比较容易理解，但是自然语言存在着巨大缺陷，例如很难表述算法中的多分支结构和循环结构。首先，自然语言很容易造成二义或歧义，例如“打死老虎”，既可以理解为“打死”的是“老虎”，也可以理解为“打”的是“死老虎”；其次，在人际交往过程中，自然语言中的语气、停顿、场景等，都将导致同一句话产生不同的理解效果，所以自然语言不适合用于表示需要精确描述的算法。

2.3.2 使用传统流程图描述算法

使用图形符号表示算法显然比使用自然语言描述算法更容易理解和使用，正所谓“千言万语不如一张图”。

1. 图形符号

传统流程图可直观地描述一个计算问题的求解步骤，主要图形符号如图 2-2 所示。

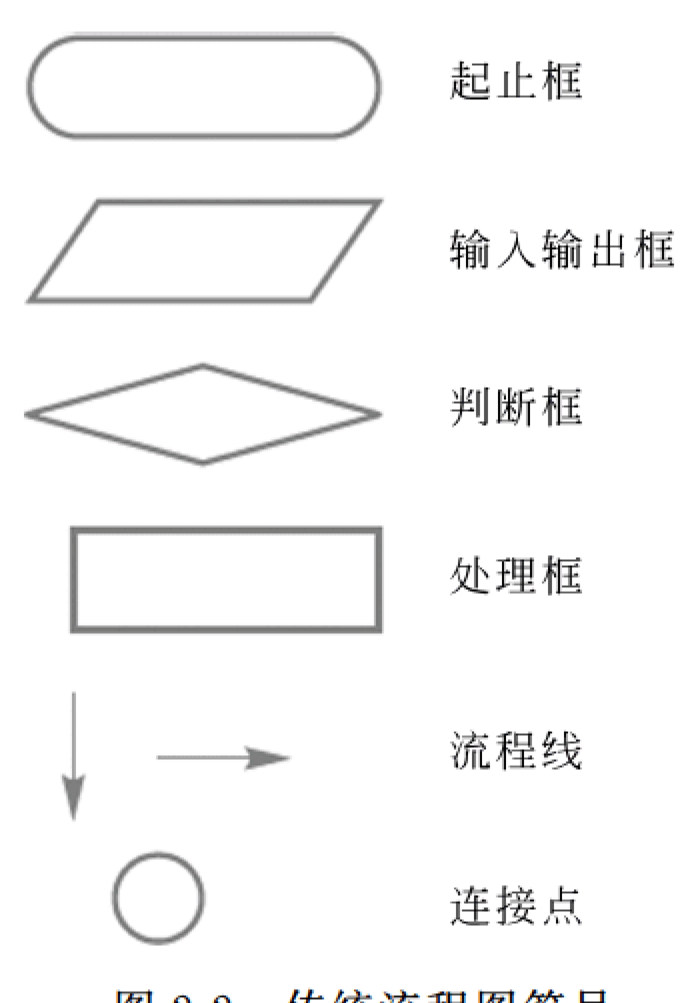


图 2-2 传统流程图符号

说明：

(1) 圆角矩形表示算法的“开始”和“结束”。

(2) 平行四边形表示输入操作和输出操作。

(3) 菱形表示表示条件判断。

(4) 直角矩形表示算法中的具体操作。

(5) 箭头表示算法的操作流程。

(6) 圆圈表示连接其他流程图的符号，具有汇合的功能。

2. 传统流程图

传统的流程图有如下 3 种结构共 5 种框图。

(1) 顺序结构。顺序结构如图 2-3 所示。

(2) 选择结构。选择结构如图 2-4 所示。

(3) 循环结构。循环结构如图 2-5 所示。

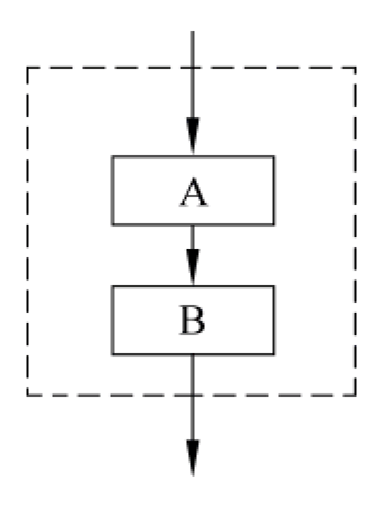
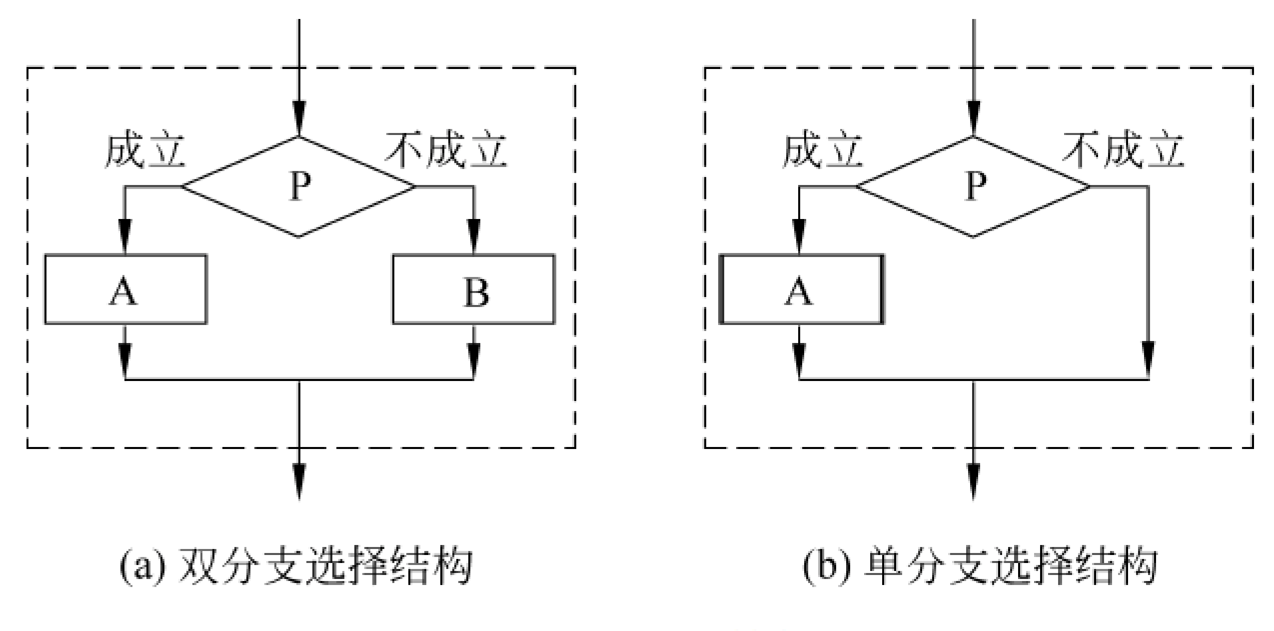


图 2-3 顺序结构

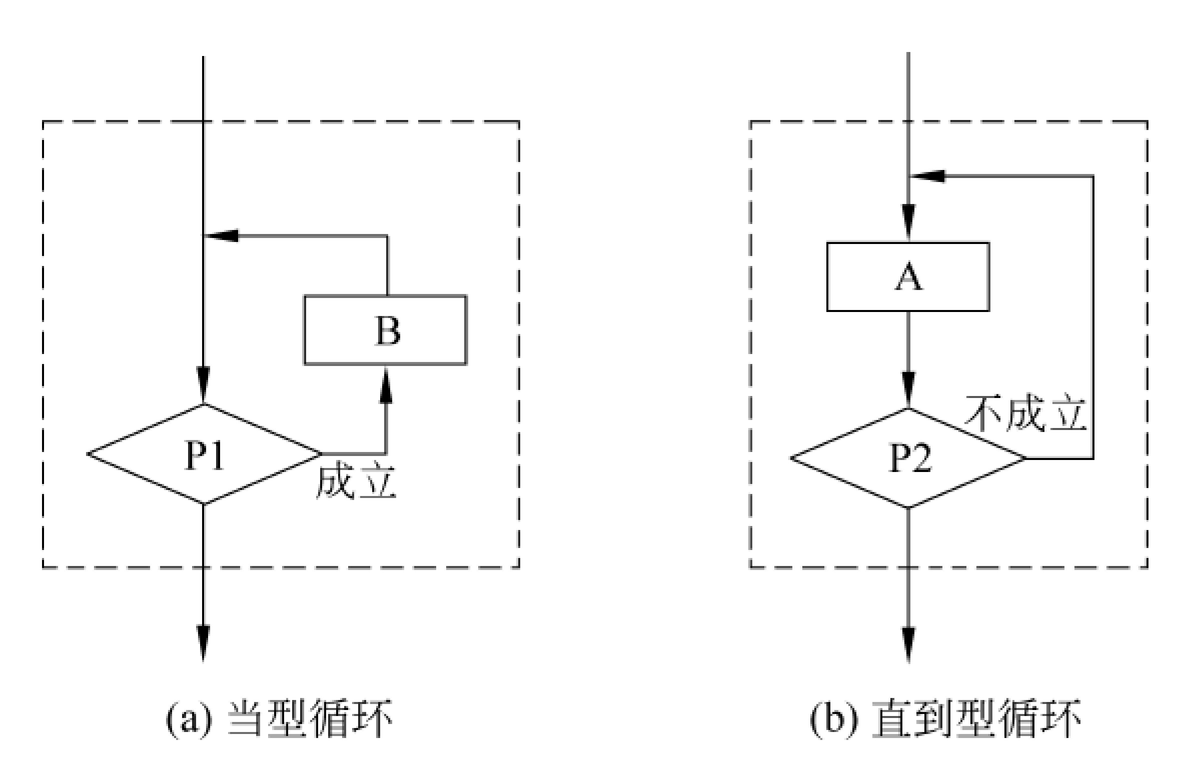
① 传统流程图的主要优点：形象直观，各种操作容易理解，也不会产生二义或歧义，算法出错时容易发现并修改。

② 传统流程图的主要缺点：所占篇幅较大且不易绘制，由于使用流



(a) 双分支选择结构　　(b) 单分支选择结构

图 2-4　选择结构



(a) 当型循环　　(b) 直到型循环

图 2-5　循环结构

程线导致算法过于灵活，不受限制，常使流程转向混乱，最终造成程序的阅读和修改困难，更不利于结构化程序的具体实现。

【例 2-4】 使用传统流程图描述阶加算法。

算法描述如图 2-6 所示。

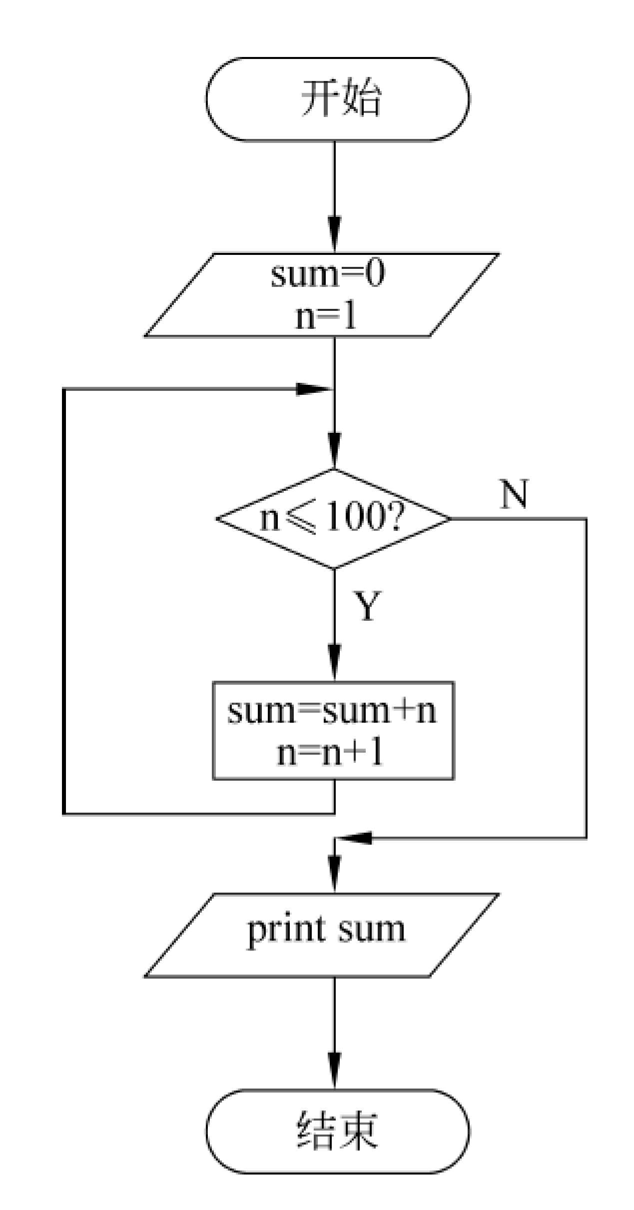


图 2-6　使用传统流程图描述阶加算法

2.3.3 使用 N-S 图描述算法

N-S 图是 Nassi 和 Shneiderman 于 1973 年提出，N 和 S 就是这两位原创者姓名的缩写，N-S 图中没有流程线，这正好符合结构化程序设计方法的要求。算法中的每个步骤都用一个矩形框来描述，且所有矩形框从上到下按执行顺序连接起来。

N-S 图符号有如下 3 种结构共 5 种框图。

1. 顺序结构

顺序结构如图 2-7 所示。

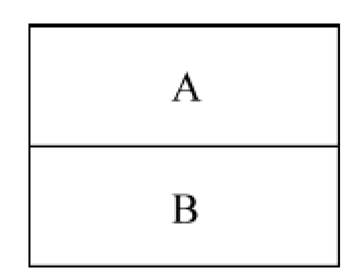


图 2-7 顺序结构

2. 选择结构

选择结构如图 2-8 所示。

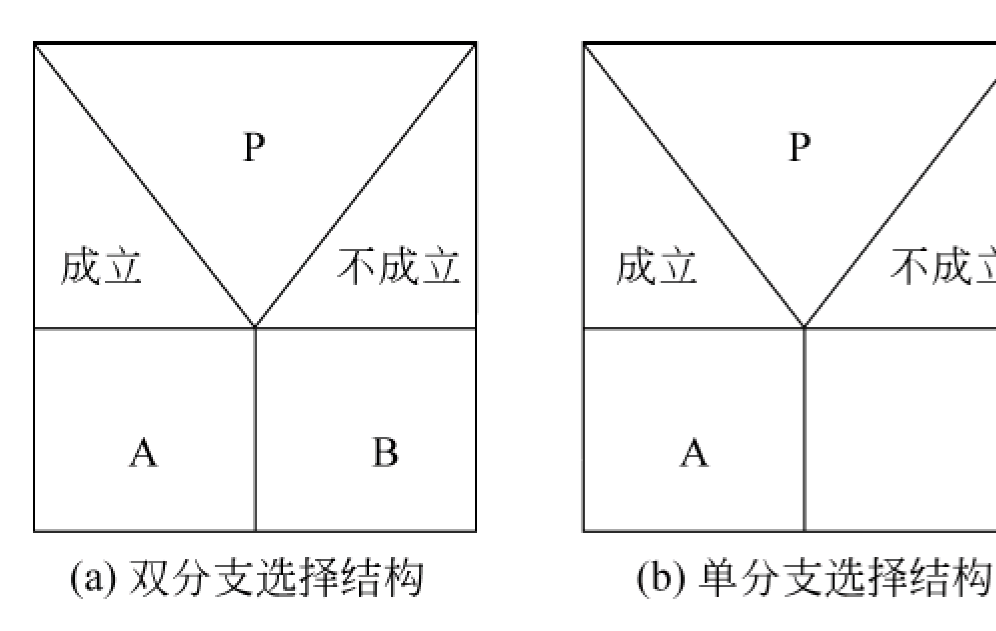


图 2-8 选择结构

3. 循环结构

循环结构如图 2-9 所示。

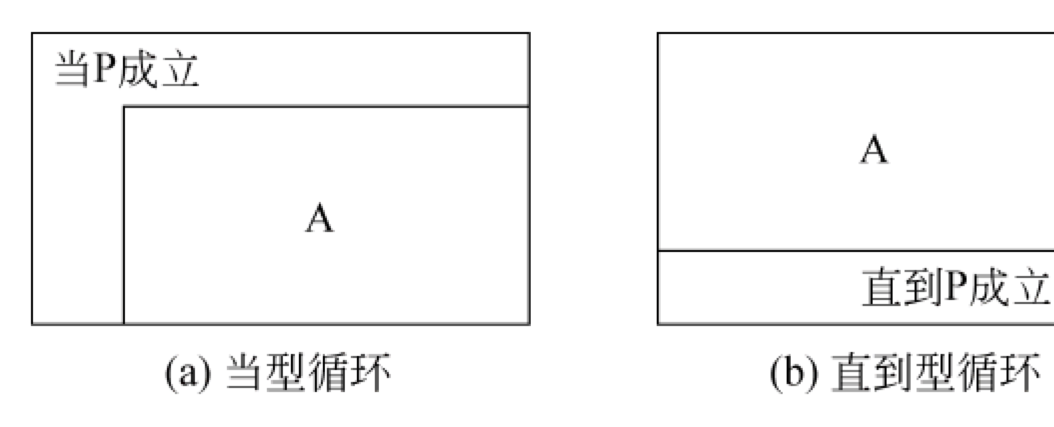


图 2-9 循环结构

在 N-S 图中，每个操作步骤都用一个矩形表示的，矩形内部可以是一条或若干条操作。在需要时，一个矩形内部还可以嵌套另一个矩形，嵌套深度并没有限制。由于 N-S 图隐含使用从上到下的操作顺序，所以整个 N-S 图只能有一个入口和一个出口。这样将限制操作过程的随意性，保证程序的良好结构，从而增加可读性。

① N-S 图的主要优点：简单且易学易用，具有较好的可读度，尤其适合描述循环和条件结构的算法。另外，N-S 图的设计意图易理解，从而为编程、查错、选择测试用例、软件维护

等提供方便。

② N-S 图的主要缺点：不容易进行手工修改，在嵌套过多时不容易绘制。

【例 2-5】 使用 N-S 图描述阶加算法。

使用 N-S 图描述算法如图 2-10 所示。

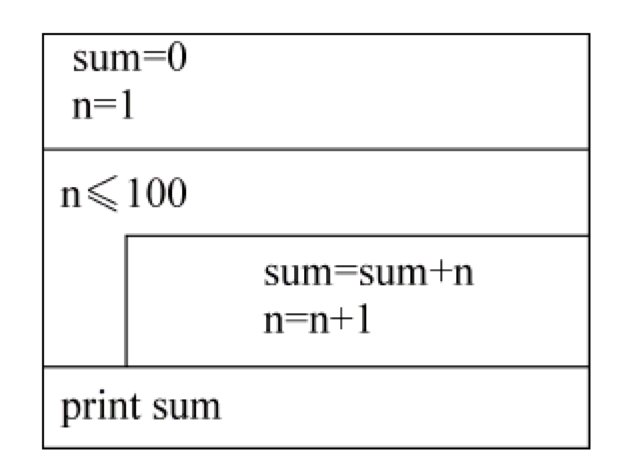


图 2-10 使用 N-S 图描述阶加算法

2.3.4 使用伪代码描述算法

伪代码(Pseudo)是一种用来描述算法时所用的非正式、透明的表述方法，属于编程描述语言。伪代码通常采用类似于自然语言与数学语言的符号体系来描述算法的操作步骤，同时使用计算机高级语言（例如 Java、Python、C、C++、Visual Basic 等）的控制结构来描述算法步骤的执行顺序。

【例 2-6】 使用伪代码描述阶加算法。

算法描述如下：

```
begin
sum=0
n=1
while (n<=100) {
    sum=sum+n
    n=n+1
}
print sum
end
```

说明：这里的符号“＝”表示赋值运算，即将右侧表达式的计算结果送到左侧变量中，符号“＜＝”就是小于等于的意思。

2.4 常用算法介绍

下面使用 N-S 图来描述算法，例如交换两个变量内容、取绝对值、阶加、阶乘、求斐波那契数列、求最大公约数、判断素数等。

2.4.1 简单算法

下面说明两个简单算法，交换两个变量的内容和取绝对值。

【例 2-7】 交换两个变量的内容。

求解方法：使用 3 个变量 a、b、t，其中变量 a 和 b 的值由键盘输入，变量 t 作为暂存单元，通过 3 次赋值操作完成交换。

使用 N-S 图描述算法如图 2-11 所示。

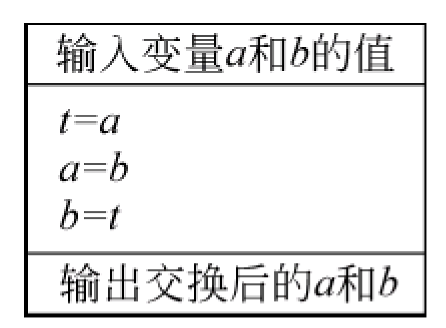


图 2-11　两个变量内容的交换

请读者自行实现，不用暂存单元，通过加减运算实现两个变量内容的交换。

【例 2-8】 取绝对值。尽管取绝对值是极简单的运算，但计算机本身并没有提供，所以只能编程实现。

求解方法：使用一个变量 n，并使用选择结构进行判断。若变量 $n \geqslant 0$，则输出 n 的值，否则输出 $-n$ 的值。

使用 N-S 图描述算法如图 2-12 所示。

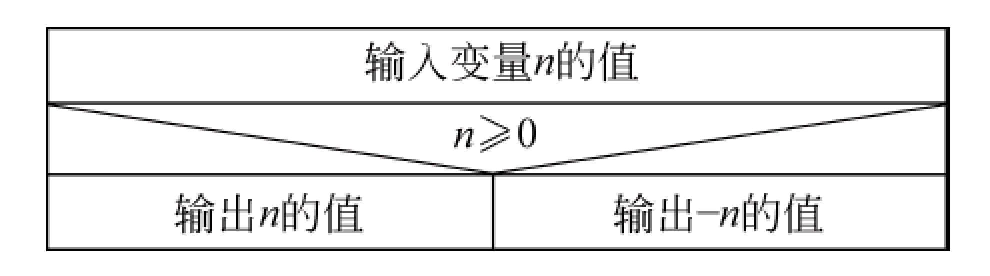


图 2-12　取绝对值

2.4.2　阶乘算法

【例 2-9】 计算 10!。

求解方法：使用循环结构来表示 10! 运算。其中，使用两个变量 f 和 n，变量 f 表示阶乘变量，初始值为 1，每次乘一个“乘数”；变量 n 表示“乘数”，初始值为 1，取值范围是 1～11，为阶乘变量 f 准备数据，而变量 n 的值为 11 时表示退出循环。

使用 N-S 图描述算法如图 2-13 所示。

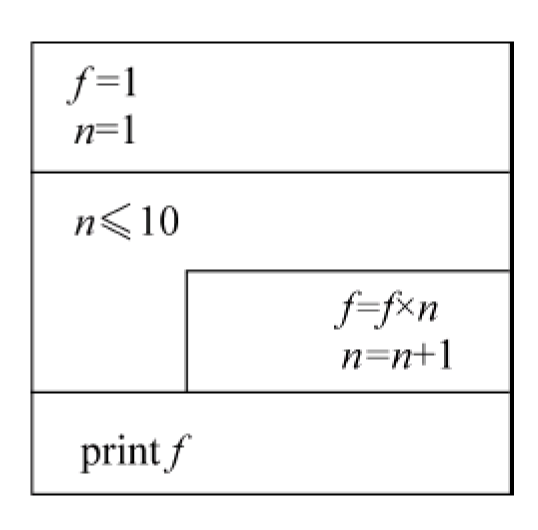


图 2-13　计算 10!

阶乘运算可以推广到其他需要连乘的算法中。

2.4.3　求斐波那契数算法

【例 2-10】 求斐波那契数列中的前 10 个数。

斐波那契数列是指：0,1,1,2,3,5,8,13,21,34,…,数学定义如下：

$f(0)=0$

$f(1)=1$

$f(n)=f(n-1)+f(n-2)$,整数 $n\geqslant 2$

求解方法：使用循环结构和迭代算法。其中,循环控制变量 k 在 2～10 范围内取值,在取值为 2～9 时求斐波那契数,取值为 10 时结束循环;另外设置 3 个变量为 f_1,f_2,f,其中 f_1的初始值为 0,f_2的初始值为 1,f 的值是 f_1 和 f_2的和,迭代过程如下：

$f=f_1+f_2$

$f_1=f_2$

$f_2=f$

使用 N-S 图描述算法如图 2-14 所示。

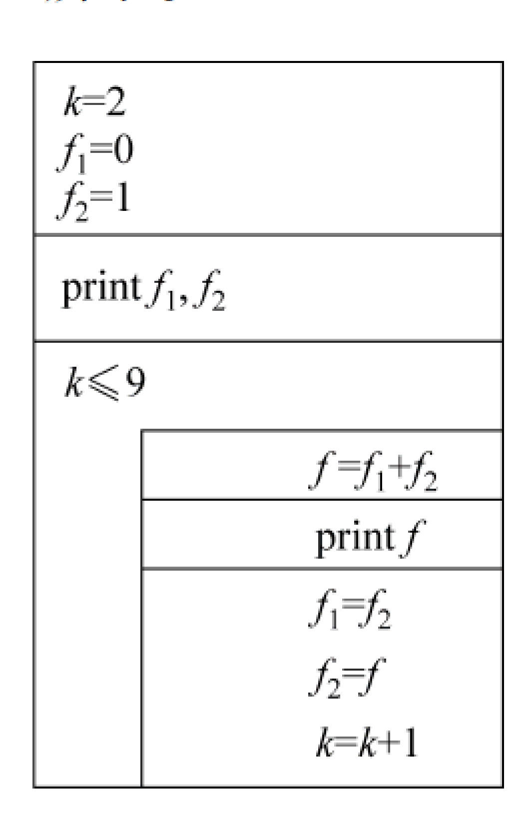


图 2-14　求斐波那契数列

2.4.4　求最大公约数算法

【例 2-11】 求两个正整数的最大公约数。

求解方法：公约数是一个能被若干个整数同时整除的整数,最大公约数指两个正整数共有约数中的最大数。下面介绍欧几里得算法(又称为辗转相除法),操作步骤如下：

步骤 1,输入两个正整数,并放入变量 m 和 n 中;

步骤 2,求出 m 除以 n 的余数放入变量 r 中;

步骤 3,若 r 为 0,则执行步骤 7,否则执行步骤 4;

步骤 4,操作 $m=n$,$n=r$;

步骤 5,求出 m 除以 n 的余数放入变量 r 中;

步骤 6,执行步骤 3;

步骤 7,n 就是所求的结果,输出结果。

使用 N-S 图描述算法如图 2-15 所示。

思考：循环操作过程是如何结束的?

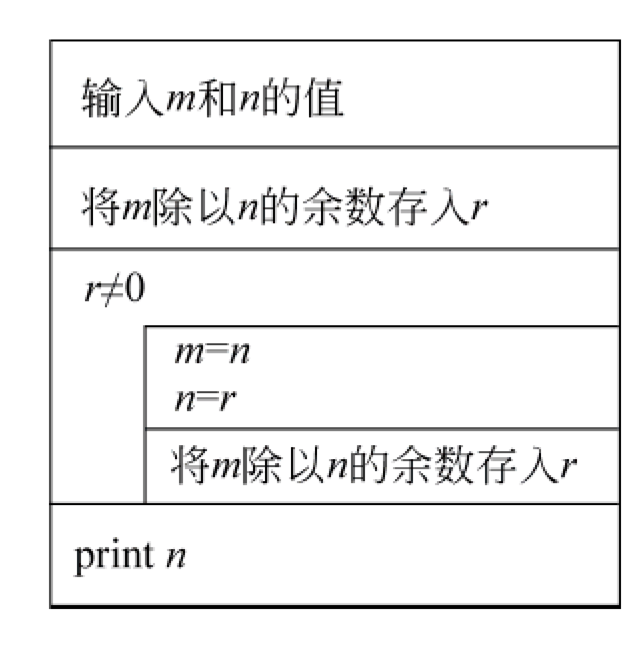


图 2-15　求最大公约数

2.4.5 判断素数算法

【例 2-12】 判断素数。素数是一个大于 1 的自然数，除了 1 和它本身外，不能被其他自然数整除，换言之就是素数除了 1 和它本身以外不再有其他的因数。

求解方法：使用穷举算法。由键盘输入一个大于 1 的自然数 n，使用一个变量 k，其取值范围从 $2\sim n-1$，用循环结构来判断所有的 k 值是否是 n 的因数。若有因数，则输出不是素数的信息并结束算法；若没有任何因数，则输出是素数的信息再结束算法。

使用 N-S 图描述算法如图 2-16 所示。

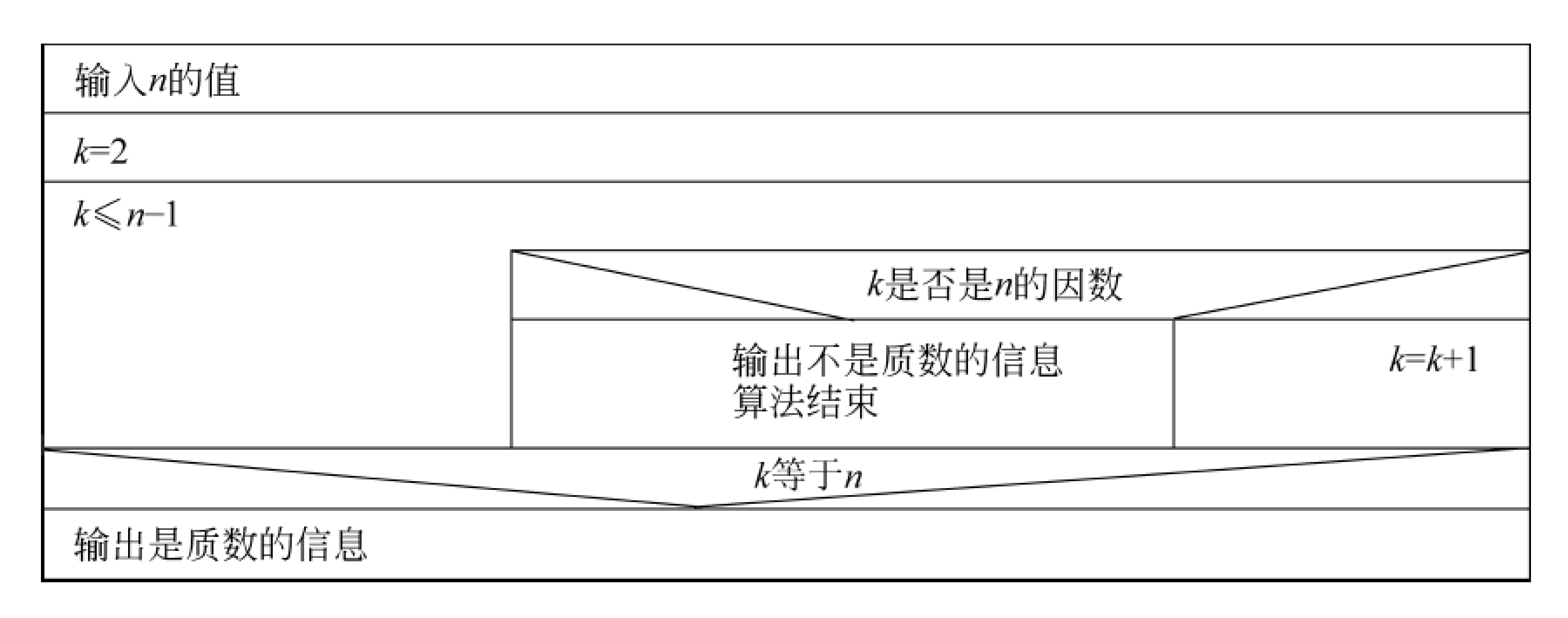


图 2-16 判断素数

2.5 综合算法介绍

下面的综合类算法包括数值计算、穷举算法、查找算法和排序算法。

2.5.1 数值计算

计算工具的发展过程体现了人类对数学计算的不懈追求，其中电子计算机做出的贡献最大。这里的数值计算就是利用计算机求解各种数学问题，并考虑误差、收敛性和稳定性等问题。下面通过两个例子进行说明。

【例 2-13】 根据公式 $e=1+1+1/2!+1/3!+\cdots+1/n!+\cdots$，计算超越数 e 的值，精确到小数位的第 6 位。

题目分析：写出计算通项的迭代公式是 $t_i=t_{i-1}/i$，其中 $i\geqslant 2$。由于循环次数不能确定，所以只能设置结束条件为最后 1 项的值小于 10^{-7}。

求解方法：使用阶加算法。使用 3 个变量，其中变量 i 用于循环控制，取值从 2 开始，其后通过加 1 增值；变量 t 表示通项，初值为 1；变量 e 用于记录每次的阶加结果，初值为 2。用循环结构来判断 t 值是否大于 10^{-7}，若大于则将 t 的值将阶加至变量 e 中并进入下次循环，否则退出循环并输出结果。

算法描述如下：

步骤 1，分别设定 i 的初值为 2，t 的初值为 1，e 的初值为 2；

步骤 2，若 $t>10^{-7}$，则执行步骤 3，否则转出执行步骤 7；

步骤 3，计算 t 除以 i 的值并赋给 t；

步骤 4,计算 e 加 t 的值并赋给 e;

步骤 5,计算 i 加 1 的值并赋给 i;

步骤 6,转去执行步骤 2;

步骤 7,输出 e 的值;

步骤 8,算法结束。

【例 2-14】 根据公式 $\sin x=x-x^3/3!+x^5/5!-x^7/7!+x^9/9!+\cdots+x^{99}/99!$,计算正弦值。

题目分析:写出计算通项的迭代公式 $t=t/[i\cdot(1-i)]$(并切换正负符号),奇数 $i\geqslant3$。由于循环次数是确定的,所以可以使用计数型的循环。

求解方法:使用阶加算法。由键盘输入数据到 x,使用变量 i,其取值范围为 3~99 中的奇数,阶加变量 s 初值为 x,变量 t 表示通项,初值为 x。使用计数型循环实现阶加过程,首先计算通项 t 的迭代值,然后将其加到阶加变量 s 中。循环结束后并输出结果。

算法描述如下:

步骤 1,由键盘输入数据到 x 中;

步骤 2,分别设定 i 的初值为 3,s 的初值为 x,t 的初值为 x;

步骤 3,若 $i\leqslant99$,则执行步骤 3,否则转出执行步骤 8;

步骤 4,计算 t 除以 $i\cdot(1-i)$ 并赋给 t;

步骤 5,计算 s 加 t 的值并赋给 s;

步骤 6,计算 i 加 2 的值并赋给 i;

步骤 7,转去执行步骤 3;

步骤 8,输出 s 的值;

步骤 9,算法结束。

思考:奇数项相加与偶数项相减是如何实现的。

2.5.2 穷举算法

穷举法又称为枚举法,这类算法将充分利用计算机语言的循环结构。基本思想是根据问题的条件确定求解问题的大致范围,并在该范围内对其进行穷举并验证,直到问题得到求解时为止。

【例 2-15】 求解百鸡问题算法。已知公鸡 5 元 1 只,母鸡 3 元 1 只,小鸡 1 元 3 只,要求用 100 元购得 100 只鸡。

题目分析:列出三元一次方程组为

$$\begin{cases}\text{cock}+\text{hen}+\text{chicken}=100\\ \text{cock}\times5+\text{hen}\times3+\text{chicken}/3=100\end{cases}$$

这是只有两个方程的三元一次方程组,即不定方程,其解集是无穷大的。但现在现实中,鸡数是整数且极为有限,所以可以利用穷举算法来找出有限的几组解。

求解方法:这是一个三重循环程序,外层循环控制变量 cock 取值为 1~19,中层循环控制变量 hen 取值为 1~32,内层循环控制变量 chick 取值为 3~99 且增量只能为 3,对其进行穷举并判断。若找出正确结果,则将其输出。

算法描述如下:

步骤 1，设定 cock 的初值为 1；

步骤 2，若 cock≤19，则执行步骤 3，否则转出执行步骤 16；

步骤 3，设定 hen 的初值为 1；

步骤 4，若 hen≤32，则执行步骤 5，否则转出执行步骤 14；

步骤 5，设定 chicken 的初值为 3；

步骤 6，若 chicken≤99，则执行步骤 7，否则转出执行步骤 12；

步骤 7，计算 cock＋hen＋chicken 并赋给 number，cock×5＋hen×3＋chicken/3 并赋给 cost；

步骤 8，若 number 和 cost 的值均等于 100，则执行步骤 9，否则转出执行步骤 10；

步骤 9，显示 cock、hen 和 chicken 的值；

步骤 10，计算 chicken 加 3 的值并赋给 chicken；

步骤 11，转去执行步骤 6；

步骤 12，计算 hen 加 1 的值并赋给 hen；

步骤 13，转去执行步骤 4；

步骤 14，计算 cock 加 1 的值并赋给 cock；

步骤 15，转去执行步骤 2；

步骤 16，算法结束。

这个算法的循环次数是 20064（即 19×32×33），可以进行优化，例如直接计算出小鸡数，从而使循环次数变成 608（19×32）。

修改后的算法描述如下：

步骤 1，设定 cock 的初值为 1；

步骤 2，若 cock≤19，则执行步骤 3，否则转出执行步骤 13；

步骤 3，设定 hen 的初值为 1；

步骤 4，若 hen≤32，则执行步骤 5，否则转出执行步骤 11；

步骤 5，计算 100－cock－hen 的值并赋给 chicken；

步骤 6，计算 cock×5＋hen×3＋chicken/3 的值并赋给 cost；

步骤 7，若 cost 的值等于 100，则执行步骤 8，否则转出执行步骤 9；

步骤 8，显示 cock，hen 和 chicken 的值；

步骤 9，计算 hen 加 1 的值并赋给 hen；

步骤 10，转去执行步骤 4；

步骤 11，计算 cock 加 1 的值并赋给 cock；

步骤 12，转去执行步骤 2；

步骤 13，算法结束。

【例 2-16】 给出求 10000 以内完全数的求解算法。完全数又称完备数，它与所有除本身外因数的和恰好相等。例如 6＝1＋2＋3，6 就是完全数。

求解方法：求解过程中将使用双重循环。其中，外层循环中的循环控制变量 n 的取值范围是 2～10000，用于穷举可能的全部数。内层循环中的循环控制变量 k 的取值范围是 1～n－1，若 k 是 n 的因数，则阶加到变量 s 中；变量 s 用于表示因数和，其初值为 0。内层循环结束后，检查变量 s 是否恰好等于 n，若相等，则输出完全数。

算法描述如下：

步骤 1，设定 n 的初值为 1；

步骤 2，若 $n \leqslant 10000$，则执行步骤 3；否则转出执行步骤 11；

步骤 3，分别设定 s 的初值为 0，k 的初值为 1；

步骤 4，若 $k < n$，则执行步骤 5；否则转出执行步骤 8；

步骤 5，若 k 是 n 的因数，则执行步骤 6；否则转出执行步骤 7；

步骤 6，计算 s 加 k 的值并赋给 s；

步骤 7，转去执行步骤 4；

步骤 8，若 s 等于 n，则显示完全数 n；

步骤 9，计算 n 加 1 的值并赋给 n；

步骤 10，转去执行步骤 2；

步骤 11，算法结束。

2.5.3 查找算法

查找是在大量数据中找出一个特定的数据。常用的查找是顺序查找和折半查找，显然顺序查找的效率太低，而折半查找（又称为二分查找）要求数组有序，是一种效率较高的查找方法。下面分别进行说明。

1. 顺序查找

【例 2-17】 在一维数组 a[n]中顺序查找是否有指定值 d，若有需给出相应的下标位置，否则给出没有找到的提示信息。

算法思想：从数据中的第一个元素开始，逐个与给定值 d 进行比较是否相等。若相等则表示查找成功，给出所找到数据的下标位置并退出循环；否则继续查找，若直到最后一个元素都没有与给定值 d 相等，则表示数据中没有所要查找的 d，即查找失败，并给出没有找到的提示信息。

求解方法：从键盘输入数据到 d 中，并初始化无序的一维数组 a[n]。求解过程中将应用到计数型循环。循环控制变量 k 取值范围是 $0 \sim n-1$，用于穷举数组 a[n]中的全部下标（以便对应数组元素）。逐个将 a[k]与给定值 d 进行相等比较。若相等则表示查找成功，显示所找到数据的下标位置并退出循环；否则继续查找，直到最后一个元素都进行比较完为止。若变量 $k \geqslant n$ 则表示数组中没有所要查找的 d，即查找失败，并给出没有找到的提示信息。

算法描述如下：

步骤 1，初始化无序的一维数组 a[n]，并输入数据到 d 中；

步骤 2，设定 k 的初值为 0；

步骤 3，若 $k < n$，则执行步骤 4，否则转出执行步骤 7；

步骤 4，若 a[k]等于 d，则显示查找成功并转出执行步骤 8；

步骤 5，计算 k 加 1 的值并赋给 k；

步骤 6，转去执行步骤 3；

步骤 7，显示查找失败；

步骤 8，算法结束。

注意：通常计算机中的下标位置是从 0 开始开始编号的。其次，这个算法分别有查找成功和失败的两个出口，这是不符合结构化程序设计原则的。

2. 折半查找

【例 2-18】 在有序的一维数组 a[n]中折半查找是否有指定值 d，若有则给出其下标位置，否则给出没有找到的提示信息。

算法思想：首先，将数组 a[n]的居中元素与指定值 d 进行比较，若两者相等表示查找成功，则显示其下标位置并退出循环；否则以居中元素为界将数组折半分成左、右两部分，如果居中元素大于指定值 d，则进一步查找左半部分，否则进一步查找右半部分。重复以上过程，若找到满足条件的 d，则表示查找成功；如果没有，则表示查找失败。

在图 2-17 中，给出折半查找过程中的变量表示情况。

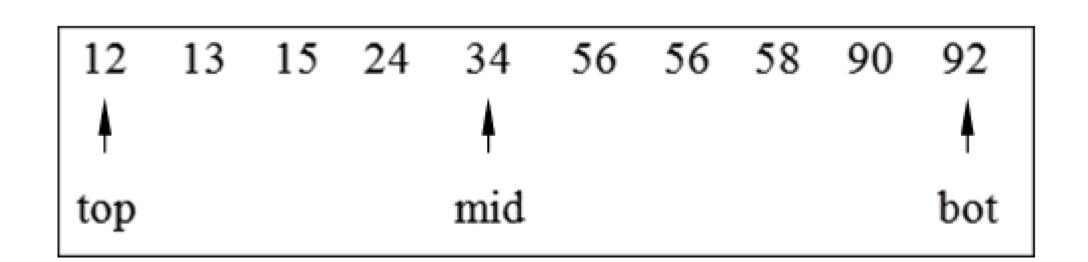


图 2-17　查找过程中的变量取值

求解方法：从键盘输入数据到 d 中，并初始化有序的一维数组 a[n]。设置变量并初始化如下：top＝0，bot＝ n－1。使用条件型循环，循环条件是 top≤bot，在循环体中首先计算 mid＝(top＋bot)/2，然后比较 a[mid]与指定值 d 是否相等，若相等则显示其下标位置并退出循环，否则将以居中元素为界将数组折半分成左、右两部分。其中左部分需修改变量为 bot＝mid－1，右部分需修改变量为 top＝mid＋1，继续循环。

算法描述如下：

步骤 1，初始化有序的一维数组 a[n]，并输入数据到 d 中；

步骤 2，分别设定 top 的初值为 0，bot 的初值为 n－1；

步骤 3，若 top≤bot，则执行步骤 4，否则转出执行步骤 11；

步骤 4，计算(top＋bot)/2 的值并赋给 mid；

步骤 5，若 a[mid]与 d 相等，则执行步骤 6，否则转出执行步骤 7；

步骤 6，显示其下标位置并转出执行步骤 12；

步骤 7，若 a[mid]＜d，则执行步骤 8，否则转出执行步骤 9；

步骤 8，计算 mid 加 1 的值并赋给 top，并转出执行步骤 10；

步骤 9，计算 mid 减 1 的值并赋给 bot；

步骤 10，转出执行步骤 3；

步骤 11，显示查找失败；

步骤 12，算法结束。

折半查找法充分利用数组有序排列的关系，采用计算思维中的分治策略，即每次将数组分为规模为一半的小数组，从而使查找次数减少。例如，若在 1000 个数据中进行折半查找，则查找次数最多为 10。

2.5.4　排序算法

无序数据的可用度非常低，尤其是在各种信息检索系统中，这在以上的顺序查找算法中

就已经得到验证。所以，目前发展出许多排序算法，例如选择排序、冒泡排序、插入排序、希尔排序、堆排序、快速排序、归并排序、基数排序、外部排序等。下面分别介绍选择排序和冒泡排序。

1. 选择排序

【例 2-19】 使用选择排序算法将无序的一维数组 a[n]中排列成从小到大的顺序。

算法思想：在要排序的一组数组中，选出最小的一个元素与第一个位置的元素进行交换；然后在剩下的元素当中再找最小的一个元素与第二个位置的元素进行交换，依次类推，直到最后两个元素进行比较完为止。

求解方法：完整的选择排序过程需要 $n-1$ 趟，操作过程如下：

第 1 趟，从 n 个元素中找出最小元素与第一个元素进行交换；

第 2 趟，从第 2 个元素开始的 $n-1$ 个元素中再选出次小元素与第二个元素进行交换；

…

第 i 趟，从第 i 个元素开始的 $n-i+1$ 个元素中选出最小元素与第 i 个元素进行交换。

直到第 $n-1$ 趟选出其中的最小元素与第 $n-1$ 个元素进行交换后，整个数组才排列有序。

注意：整个选择排序完全是自动化的，即使在中间某趟处理完成后，数组已经排列有序，其排序过程还会“冗余”地继续进行下去。

算法描述如下：

步骤 1，初始化一维数组 a[n]；

步骤 2，设定 i 的初值为 0；

步骤 3，若 $i<n-1$，则执行步骤 4，否则转出执行步骤 12；

步骤 4，设定 min 的初值为 i，j 的初值为 $i+1$；

步骤 5，若 j<n，则执行步骤 6，否则转出执行步骤 9；

步骤 6，若 a[j]<a[min]，则将 j 的值赋给 min；

步骤 7，计算 j 加 1 的值并赋给 j；

步骤 8，转去执行步骤 5；

步骤 9，将 a[i]和 a[min]的内容交换；

步骤 10，计算 i 加 1 的值并赋给 i；

步骤 11，转去执行步骤 3；

步骤 12，显示有序数组 a[n]；

步骤 13，算法结束。

说明：由于算法中需要将 a[i]和 a[min]的内容交换，所以最小元素的下标也要保存下来，故算法中并没有直接找出最小元素。

2. 冒泡排序

【例 2-20】 使用冒泡排序算法将无序的一维数组 a[n]中排列成从小到大的顺序。

算法思想：在要排序的一组数组中，对当前尚未排序的全部元素，自上而下对相邻的两个元素依次进行比较和调整位置，让较大元素往下沉，较小元素往上冒，即将相邻元素进行交换。第一趟冒泡过程将使最大元素沉底，其后缩小范围再进行下一趟，直到进行 $n-1$ 趟为止。与选择排序一样，也会出现“冗余”比较问题。

冒泡排序过程的示例如图 2-18 所示。

初始数	第 1 趟	第 2 趟	第 3 趟	第 4 趟	第 5 趟	第 6 趟	第 7 趟
36	32	32	32	32	12	12	12
32	36	36	36	12	32	32	32
88	68	35	12	35	35	35	35
68	35	12	35	36	36	36	36
35	12	68	63	63	63	63	63
12	74	63	68	68	68	68	68
74	63	74	74	74	74	74	74
63	88	88	88	88	88	88	88

图 2-18　冒泡排序

求解方法：使用双重循环。其中，外层循环控制变量 i 的取值范围是 $0 \sim n-2$，用于表示冒泡过程的趟数；内层循环控制变量 j 的取值范围是 $i \sim n-2$，用于表示相邻元素是否交换的处理。

算法描述如下：

步骤 1，初始化一维数组 a[n]；

步骤 2，设定 i 的初值为 0；

步骤 3，若 $i < n-1$，则执行步骤 4，否则转出执行步骤 11；

步骤 4，设定 j 的初值为 i；

步骤 5，若 $j < n$，则执行步骤 6，否则转出执行步骤 9；

步骤 6，若 a[j]>a[$j+1$]，则将 a[j]和 a[$j+1$]的内容交换；

步骤 7，计算 j 加 1 的值并赋给 j；

步骤 8，转去执行步骤 5；

步骤 9，计算 i 加 1 的值并赋给 i；

步骤 10，转去执行步骤 3；

步骤 11，显示有序数组 a[n]；

步骤 12，算法结束。

2.6　迭代、递推和递归

迭代、递推和递归均属于循环结构，但实现方式完全不同，下面分别进行介绍。

2.6.1　迭代

由于计算机具有运算速度快、适合自动进行重复操作的特点，这就让迭代算法成为利用计算机求解问题的一种基本方法。具体操作是让计算机对一定步骤进行重复执行，在每次执行这些步骤时，都从变量的原值推出它的一个新值，最终逼近结果。当然，迭代算法也应该具有适当的收敛性，即能够在复杂度较低(例如在 1×10^6 次迭代或 10s 以内)的情况下能够得到结果。

【例 2-21】 验证角谷猜想。猜想的大意是，日本著名学者角谷静夫在 1930 年提出，设

定正整数 n，若 n 为偶数，则就将它变为 $n/2$，否则将它变为 $3n+1$。不断重复这样的运算，经过有限步转换后，一定可以得到 1。人类在超过 80 年中通过大量的验算，从来没有发现反例，当然也没有人能够证明这是定理。

以正整数 6 为例说明重复运算的过程如下：

6→6/2→3→3×3+1→10→10/2→5→5×3+1→16→16/2→8→8/2→4→4/2→2→2/2→1

算法思想：首先设定角谷猜想的验证过程不可能出现无穷循环，然后经过有限次迭代（即循环）就能够得到 1，从而结束验证过程。

求解方法：从键盘输入一个正整数到 n 中。重复进行循环条件判断，若 n 为 1 则结束循环，否则做循环体，在循环体内判断 n 为偶数时，将其修改为 $n/2$，否则修改为 $3n+1$。

算法描述如下：

步骤 1，从键盘输入一个正整数到 n 中；

步骤 2，若 $n>1$，则执行步骤 3，否则转出执行步骤 7；

步骤 3，若 n 为偶数，则执行步骤 4，否则转出执行步骤 5；

步骤 4，计算 n 除以 2 的值并赋给 n 并转出执行步骤 6；

步骤 5，计算 $3n+1$ 的值并赋给 n；

步骤 6，显示 n，并转去执行步骤 2；

步骤 7，算法结束。

2.6.2 递推

与迭代算法类似，递推算法充分利用计算机具有运算速度快、适合自动进行重复操作的特点。递推算法是将一个复杂的计算过程转化为一个（若多个）简单过程的重复计算过程，即按照一定的规律计算序列中的每一项，通常是由计算前面的一些项来计算出序列中的指定项或后一项。当然，递推算法也应该具有适当的收敛性，即能够在复杂度较低的情况下能够得到结果。

【例 2-22】 构造斐波那契数列的前 30 个数。

算法思想：斐波那契数列为 1，1，2，3，5，8，13，21，34，55，89，144，…。如果设 $f(n)$ 为该数列的第 n 项，并且 $f(1)$、$f(2)$ 的初值均为 1，则 $f(n)=f(n-1)+f(n-2)$，显然这是一个线性递推数列。

求解方法：在初始值确定后，直接利用循环结构和递推公式可以求出全部斐波那契数。递推公式为 $f=f_1+f_2$、$f_1=f_2$ 和 $f_2=f$。

算法描述如下：

步骤 1，从键盘输入数据（取 30）到 n 中；

步骤 2，分别设定 f_1 和 f_2 的初值均为 1，i 的初值为 3；

步骤 3，若 $n\leqslant 30$，则执行步骤 4，否则转出执行步骤 8；

步骤 4，计算 f_1 加 f_2 的值并赋给 f，并显示 f；

步骤 5，继续递推，将 f_2 的值并赋给 f_1，将 f 的值并赋给 f_2；

步骤 6，计算 n 加 1 的值并赋给 n；

步骤 7，转去执行步骤 3；

步骤 8,算法结束。

2.6.3 递归

下面通过计算阶乘来介绍递归概念。

【例 2-23】 计算 $n!$。

算法思想:在进行问题求解前,首先分析分段函数

$$\begin{cases} f(n)=1 & n=1 \\ f(n)=n\cdot f(n-1), & n>1 \text{ 且 } n \text{ 为整数} \end{cases}$$

下面的函数计算过程,以正整数 10 为例进行说明。

$f(10)\to 10\times f(9)\to 90\times f(8)\to 720\times f(7)\to 5040\times f(6)\to 30240\times f(5)\to 151200\times f(4)\to 604800\times f(3)\to 1814400\times f(2)\to 3628800\times f(1)\to 3628800$

从以上示例中可以发现,在调用函数的过程中又出现调用该函数本身,这就是函数的递归调用。一个计算问题使用递归调用来求解时,必须符合以下 3 个条件。

(1) 有一个结束递归调用过程的条件。

(2) 可以运用转化过程使问题得以简化并加以解决。

(3) 可以将所求解的计算问题转化为另一个简化的计算问题,而二者之间的解法是完全相同的,被处理的对象必须有规律地递增或递减。

递归是由函数调用实现的,将在第 5 章中进行深入介绍。

习 题 2

一、简答题

1. 结构化程序设计的原则是什么?
2. 结构化程序有哪些基本结构?
3. 结构化程序有哪些特点?
4. 什么是面向对象程序设计方法?
5. 什么是对象? 什么是类?
6. 什么是算法? 简述算法的五大特征。
7. 传统流程图有哪些缺点?
8. N-S 图有哪些优点?

二、绘制 N-S 图

1. 判断一个正整数能否被 3 除尽(整除)。
2. 输入 3 个整数 x、y、z,按由小到大的顺序输出它们的值。
3. 从键盘依次输入 10 个整数,求出其中的最大数。
4. 使用穷举算法求两个正整数的最小公倍数。

三、算法描述

1. 一个整数加上 100 后是完全平方数,再加上 168 还是完全平方数,找出该数。
2. 使用公式 4×(1−1/3+1/5−1/7+⋯+1/97−1/99),计算圆周率 π 的近似值。
3. 使用公式(2/1)×(4/3)×⋯×(100/99)/4,计算圆周率 π 的近似值。

4. 计算公式 2/1＋3/2＋5/3＋8/5＋13/8＋21/13＋…的前 20 项之和。

5. 统计 1、2、3、4 的数字能组成互不相同且没有重复数字的三位数的数量。

6. 判断一个 9 位数是否为回文数。若一个自然数与其各位数字相反排列的数相等，则称为回文数，例如 1234321 就是回文数。

7. 将一个数插入到有序数组中，并保持有序。

第 3 章　数据与计算

数据和计算就是计算思维中的“计算环境”，它是一个程序运行所需要的硬件和软件。其中，数据部分包含数据类型、存储字节数量、存储表示方式与精度、数值大小与范围等；运算部分包含算术运算（含整除、求余）、关系运算、逻辑运算、位操作运算、赋值运算、条件运算等；组合符号部分包含关键字、标识符、变量名、表达式等；顺序程序部分包含常量定义、变量声明、输入输出的函数调用、模块引用、源程序的结构与书写格式、赋值语句等。

本章首先介绍输入输出操作、Python 编码风格与简单程序组成、组合符号等，然后介绍数据类型的概念，最后分别介绍数字、字符串、布尔数据、列表、元组、字典、集合等数据类型及其运算。

注意，一台计算机的计算环境是固定的，程序运行是不可能超越这个环境的。例如，不能在 32 位 Windows 系统中安装 64 位的 Python 语言，反之亦然。仅就二进制浮点数据而言，国际电气电子工程师学会早在 1985 年就制定出 IEEE 754 标准。在这个标准中定义了浮点数的格式、一些特殊数值、非数值（例如 NaN）等。通过这些工业标准，使制造软件（智力）产品的过程如同工业流水线一样，这也是国内外教育界大力推动计算思维的原因之一，毕竟，学习 Python 程序设计的目标之一是在特定计算环境下构造新的计算产品。

3.1　输入输出

大多数程序都包含输入数据和输出结果，没有输入数据的程序将运行在封闭的数据环境中，缺乏通用性，没有输出结果的程序则完全没有意义。另外，程序运行时序通常都是输入数据、进行计算和输出结果，这就是简单程序的形式。下面就介绍输入输出操作。

3.1.1　输入数据

在输入数据方面，Python 提供内置库中的 input()标准函数来实现，一般调用格式如下：

```
var=input(<prompt>)
```

其中，<prompt>是用引号界定的提示信息（即字符串）。

input()函数的功能是提示用户输入一个数据，并将该数据作为字符串送给指定的变量。

【例 3-1】　input()函数示例。

在 IDLE 交互环境中，输入如下命令：

```
>>>data=int(input("请输入一个整数数据："))
>>>data
```

运行结果如图 3-1 所示。

```
>>> data=int(input("请输入一个整数数据: "))
请输入一个整数数据: 36
>>> data
36
```

图 3-1 例 3-1 的运行结果

3.1.2 输出数据

1. 简单输出操作

在输出数据方面,Python 提供内置库中的 print()标准函数来实现,格式如下:

```
print(v1, v2, …)
```

print()函数用于显示一行内容,即显示值表(v_1,v_2,…)中的全部(可以只有一个)数据,值表中的数据间必须用逗号分隔。

【例 3-2】 print()函数示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>x=12
>>>y=34
>>>print(x, y)
```

运行结果:

```
12 34
```

2. 详解 print()函数

print()函数的一般调用格式如下:

```
print(<objects>, <sep>="", <end>="\n", <file>=sys.stdout, <flush>=False)
```

说明:

(1) ＜objects＞:表示可以一次输出多个对象,多个对象需要用逗号分隔。

(2) ＜sep＞=" ":用来间隔多个对象,默认值是一个空格,也可自行设置。

(3) ＜end＞="\n":用于设定结尾方式,默认是换行,也可自行设置。

(4) ＜file＞=sys. stdout:用于指定对象必须要有写(write)的方法。

(5) ＜flush＞:该参数取 True 表示强制清除缓存,取 False 表示缓存由文件表示。

3. 数据格式

为表示直观的显示效果,需要对不同类型数据格式进行描述,如表 3-1 所示。

表 3-1 数据格式描述

转换类型	含　　义
d	有符号位的十进制整数
o	无符号位的八进制
u	无符号位的十进制

续表

转换类型	含　　义
x 或 X	无符号位的十六进制
e 或 E	科学计数法表示的浮点数
f 或 F	十进制浮点数
g 或 G	若指数大于－4 或小于精度值，则和 e 相同，否则和 f 相同
g	单个字符(接受整数或者单字符字符串)
r	字符串(使用 repr 转换任意 Python 对象)
s	字符串(使用 str 转换任意 Python 对象)

【例 3-3】 格式输出示例。

源程序如下：

```
year=2017
month=12
day=25
#格式符%02d:数字转成两位整型，不足两位高位补 0，字符原样显示
print("整数数据:\t%04d-%02d-%02d"%(year,month,day))
pi=3.1415926
#格式符%06.2f:总宽度为 6，小数 2 位，小数点 1 位，高位补 00
print("实数数据:\t%06.2f"%pi)
#格式符%.2e:以科学计数法输出浮点型，保留 2 位小数
print("指数数据:\t%.2e"%pi)
n=128
#格式符%4d:输出十进制，高位补空格
print("十进制数据:\t%4d"%n)
#格式符%4o:输出八进制，高位补空格
print("八进制数据:\t%4o"%n)
#格式符%04x:输出两位十六进制，字母小写，高位补 00
print("十六进制数据:\t%04x"%n)
#格式符%04X:输出四位十六进制，字母大写，高位补 00
print("十六进制数据:\t%04X"%n)
```

运行结果如图 3-2 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
整数数据:        2017-12-25
实数数据:        003.14
指数数据:        3.14e+00
十进制数据:       128
八进制数据:       200
十六进制数据:     0080
十六进制数据:     0080
```

图 3-2　例 3-3 的运行结果

3.2 编码风格与简单程序

在介绍如何编写程序前，还需要介绍 Python 程序的编码风格。要让程序具有较好的可读性，养成良好的编码风格是至关重要的。

3.2.1 编码风格

作为现代语言，Python 引入大多数软件开发过程中遵循的编程风格。它给出了一个高度可读、视觉感知极佳的编码风格。每个 Python 程序员都应该理解并运用这些风格，其中的大多数要点都会对编程有所帮助。

(1) 使用 4 个空格表示缩进层次，而不是 Tab 键。

(2) 每行确保不要超过 79 个字符，行尾字符或第 80 个字符专门表示换行。

(3) 适当使用空行分隔语句块、函数、类、模块等对象。

(4) 最好让注释信息独占一行，即成为注释行。

(5) 把空格放到操作符两边以及逗号后面。

(6) 使用统一的标识符命名，例如使用著名的 SmallTalk 法则。

(7) 函数名和方法名用小写字母，并用下画线连接成组合符号。

(8) 类名首字母大写，其余使用小写字母。

(9) 程序结构要简单易读，使用标准语句模式。

3.2.2 简单程序

简单程序是指该程序自始至终按照语句序列的排列顺序，从头到尾逐条执行，即严格遵循"属性声明→准备数据→进行计算→输出结果→程序结束"的处理时序。

【例 3-4】 计算矩形面积。

源程序如下：

```
a=int(input("矩形长度:\t"))                    #输入矩形长度
b=int(input("矩形宽度:\t"))                    #输入矩形宽度
area=a * b                                     #计算矩形面积
print("矩形面积:\t",area)                       #输出结果
```

程序运行时输入矩形的长度和宽度是 5 和 8，则运行结果如图 3-3 所示。

```
====================== RESTART: D:\Python36\ch3\p1.py ======================
矩形长度:        5
矩形宽度:        8
矩形面积:         40
```

图 3-3 例 3-4 的运行结果

注意：本例中的第 1 行调用 int()函数将字符串格式的数据转换成整型数据。另外，字符\t 是表示功能转义的特殊符号，可实现左对齐。

【例 3-5】 输入一个华氏温度值，编程输出对应的摄氏温度值。

将华氏温度(用 f 表示)转换成摄氏温度(用 c 表示)的公式是 $c=5\times(f-32)/9$。

源程序如下：

```
f=float(input("华氏温度:\t"))                    #输入华氏温度值
c=5*(f-32)/9                                     #计算摄氏温度值
print("摄氏温度:\t",c)                           #输出摄氏温度值
```

程序运行时输入华氏温度数据是 100,则运行结果如图 3-4 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
华氏温度:        100
摄氏温度:         37.77777777777778
```

图 3-4　例 3-5 的运行结果

本例中包含两个整数相除(实除运算)导致结果为实型数据。另外,通常输入数据是从最左侧开始的,而这里的摄氏温度值是一个 18 位的实型(含正负符号和小数点的 float 类型)数据,这样使输入数据与输出数据没有办法实现左对齐或右对齐。

3.3　组合符号

通常计算机语言(例如 Java、C、Python 等)都是使用键盘符号作为编程所用的字符集,字符集包含大小写字母、阿拉伯数字、空格、标点、特殊字符、功能字符等。将字符集中的字符按照特定规则进行组合,就成为语句中的相关语法成分。"组合符号"包括标识符(含预定义标识符)、运算符、分隔符、字面量(即常量)等,其中标识符可以用于表示关键字、变量名、函数名、对象名、类名、模块名、包名等。

运算符还可以与常量、变量、函数调用等一起构成计算式(表达式),从而表示各种运算,通常运算符是由一个或多个字符组成的。另外,分隔符用于区分不同的语法成分,常用分隔符有逗号和空格两种。逗号主要用在类型说明和函数的参数表中,用于分隔各个变量。空格用于在语句各成分之间作为分隔符。

3.3.1　标识符

1. 标识符及其定义

在程序中使用的关键字、变量名、函数名、类名、文件名等统称为标识符。除语句定义符、关键字、标准函数名等均是由系统定义以外,其余都是由用户自行定义的。Python 规定,标识符只能是由字母(A~Z,a~z)、下画线和数字(0~9)构成的一个字符串,并且其中的第一个字符必须是字母或下画线。在 Python 语言中有许多预定义的运算符号如+、-、*、/等,它们是不可以用于定义标识符的。

表 3-2 中的标识符都是合法的。

表 3-2　合法标识符示例

标 识 符	说　　明
average_score	由字母和下画线构成
grade80	由字母和数字构成
_numbers	由字母和下画线构成并以下画线开头

表 3-3 中的组合符号都是不合法的标识符。

表 3-3　不合法的组合符号示例

标 识 符	说　　明	标 识 符	说　　明
x*2	使用非法字符 *	-ab	由减号作为首字符
2n	由数字作为首字符	$ salary	由美元符作为首字符

2. 定义标识符的要求

在定义标识符时，还应该特别注意如下要求。

(1) Python 并没有限制标识符的长度，但是任何计算环境的资源都是有限的，所以建议在使用标识符时将其长度约束在 32 个字符以内。

(2) Python 自动区别标识符中的大小写字母，例如 age 和 AGE 就是两个不同的标识符。

(3) 不要随意定义标识符，由于因为标识符是用于标识一个语法成分的符号，在命名时应尽量沿用相应的意义(例如计算习惯)，以提高程序的可读性，最好做到"见名知义"。例如，用 salary 和 pay 表示工资是比较合理的。

(4) 一些组合符号(例如 if、for 等)已经被 Python 作为关键字，不能作为标识符。

(5) 以双下画线开始和结束的组合符号通常被 Python 定义特殊含义，例如__init__为类的构造方法(函数在对象编程中的称呼)，也是不能作为标识符的。

(6) 避免使用预定义标识符名作为自定义标识符，例如 int、float、list、string、tuple 等。

3.3.2　关键字

关键字是在 Python 中具有特定意义的字符串，通常也称为保留字，即系统已经进行定义并约束其使用范围，所以用户自定义标识符不能与关键字相同。

【例 3-6】 使用 Python 帮助系统查看关键字。

具体操作如下：

进入 IDLE 交互环境中，并输入如下命令：

```
>>>help()
```

输入命令后的显示内容如图 3-5 所示。

```
>>> help()

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
```

图 3-5　help()命令

如图 3-5 所示，在左下角的显示信息表示进入 IDLE 帮助系统，此时的提示符为 help>字符串。这时要查看关键字列表，可输入如下命令：

```
help> keywords
```

输入命令后的显示内容如图 3-6 所示。

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass

help>
```

图 3-6　关键字

要退出帮助系统,可输入如下命令:

```
help> quit
```

输入命令后的显示内容如图 3-7 所示。

```
help> quit

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>>
```

图 3-7　退出帮助系统

此时的提示符还原为>>>字符串,表示再次进入 IDLE 交互环境。

3.3.3　预定义标识符

Python 语言包含许多系统内置的类名、对象名、异常名、函数名、方法名、模块名、包名等对象的预定义名称,例如 math、float、ArithmeticError、print 等,建议读者避免使用预定义标识符名作为自定义标识符。

【例 3-7】 使用 Python 的 help()内置函数查看内置函数。

要显示 input()内置函数的详细内容,可在 IDLE 交互环境中输入如下命令:

```
>>>help(input)
```

输入命令后的显示内容如图 3-8 所示。

```
>>> help(input)
Help on built-in function input in module builtins:

input(prompt=None, /)
    Read a string from standard input.  The trailing newline
is stripped.

    The prompt string, if given, is printed to standard outp
ut without a
    trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Retu
rn), raise EOFError.
    On *nix systems, readline is used if available.
```

图 3-8　函数 input()的帮助信息

【例 3-8】 使用 Python 的 help()内置函数查看内置异常。

要显示内置异常 ArithmeticError 的详细内容，可在 IDLE 交互环境中输入如下命令：

```
>>>help(ArithmeticError)
```

输入命令后的显示内容如图 3-9 所示。

```
>>> help(ArithmeticError)
Help on class ArithmeticError in module builtins:

class ArithmeticError(Exception)
 |  Base class for arithmetic errors.
 |
 |  Method resolution order:
 |      ArithmeticError
 |      Exception
 |      BaseException
 |      object
 |
 |  Methods defined here:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate
signature.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for
accurate signature.
```

图 3-9　计算出错 ArithmeticError 异常的帮助信息(部分截图)

3.3.4　命名规则

在使用组合符号时，应该尽量做到见名知意，这样可以增加程序的可读性。Python 倡导并遵循的命名规则如表 3-4 所示。

表 3-4　Python 遵循的标识符命名规则

项　　目	说　　明
文件名	全小写，可用下画线作为分隔符
包或模块	全小写，可用下画线作为分隔符，例如 mypackage 或 my_package
类	单词中的首字母大写，内部类可用额外的前导下画线，例如 Myclass
函数或方法	全小写，可用下画线作分隔，例如 my_example，my__function 等
函数或方法的参数	用 self 作实例方法的第 1 个参数，用 cls 作为类方法的第 1 个参数。若函数的参数名与保留字同名，可用后缀下画线区分
全局变量	对 from module import *，若要阻止全局变量则可在其上加前导下画线

3.4　数据类型

数据类型是指简单数据的基本属性，这是一个十分重要的概念，因为数据运算必须遵守一条基本原则：只有相同或兼容类型的数据之间才能进行运算。

3.4.1　数据类型及其分类

1. 基本数据类型与复合数据类型

基本数据类型也称为简单数据类型，这些类型的数据是不能再分解的。在这些数据类型的基础上，可以创建其他的数据类型(又称为复合数据类型)，例如列表、元组、集合、字典

等，均是由若干元素合成的。Python 中的基本数据类型包括整型、浮点型、布尔类型等，这类数据具有明确的数据范围和相应的运算模式。

2. 不可变数据类型与可变数据类型

在了解数据类型和构成后，还需要知道数据类型是否是可变的。Python 中的不可变数据类型是不允许元素的值发生变化的，但是可变数据类型则会允许元素的值发生变化，即若对此类变量进行像 append、＋、＋＝等修改操作，系统均会正常运行。

在 Python 中，可变数据类型包括整型、浮点型、布尔类型、列表和字典，不可变数据类型包括字符串型、元组等。

3.4.2 常量和变量

在介绍计算机中的数据时，自然要区分常量和变量。

1. 常量

所谓常量（又称为字面量）是指在程序运行过程中始终都不会发生变化的一个值，它完全是由书写形式确定的，而与数学、物理、化学等领域中的经典常数没有关联。在编程过程中，经常将一些关键的数据设置成常量，这样做可以提高程序的可读性，而且对程序设计的扩充与修改也是十分方便的。

2. 变量

所谓变量是指在程序运行过程中可能会发生变化的一个量，本质上是所对应存储单元的内容发生了变化。在程序设计过程中，正是由于变量值不断发生变化，才能使计算结果最终能够满足实际的需要。合理地使用变量可以提高程序的可读性，而且有利于对程序进行扩充修改。

3.5 数字数据

数字数据主要包括整数、实数、分数和复数，基于计算机数据的离散且有限的性质，这些数据作为类型分别称为整型、实型、分数型和复数型。

3.5.1 整型数据

1. 整型常量

在 Python 中，允许使用 4 种整型（int）常量，除默认的十进制常量外，还可以使用二进制、八进制、十六进制等数制表示的常量，不过要通过添加前缀来与十进制常量进行区分，二进制、八进制和十六进制数的前缀分别是 0b、0o 和 0x（其中 x 可写为 X）。

【例 3-9】 int 数据示例。

源程序如下：

```
a=1234
b=0o1234                                            #八进制数
c=0X1234                                            #十六进制数
print("a=",a,"\nb=",b,"\nc=",c)                     #转义字符\n 表示换行
```

运行结果如图 3-10 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
a= 1234
b= 668
c= 4660
```

图 3-10　例 3-9 的运行结果

从运算结果可以发现，一个由数字组成的字符串按不同进制识别，将得到不同的结果，这就导致计算机中必须严格区分各种数据类型。

2. 算术运算

整型数据的算术运算如表 3-5 所示。

表 3-5　算术运算

运算符	名称	描　　述
+	加	两个数据相加
-	减	表示负数，或一个数减去另一个数
*	乘	两个数相乘或返回一个被重复若干次的字符串
/	除	两个数据相除
//	整除	两个数据相除后的整数商部分
%	取余	两个数据相除后的余数部分
**	乘方	两个数据的乘方运算

在 Python 中，整除运算符//可以用于实数运算，例如 5.2//3 的结果是 1.0，而 5/2 的结果是 2.5。

【例 3-10】 算术运算示例。

源程序如下：

```
a=2
b=4
c=5
print("c%a=",c%a)                        #整数取余运算
print("c/a=",c/a)                        #除法运算，结果为 2.5
print("c//a=",c//a)                      #整除运算，结果为 2
print("b**a=",b**a)                      #乘方运算
print("c**a**b=",c**a**b)                #乘方运算从右至左计算为(右结合)
```

运行结果如图 3-11 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
c%a= 1
c/a= 2.5
c//a= 2
b**a= 16
c**a**b= 152587890625
```

图 3-11　例 3-10 的运行结果

【例 3-11】 在居民身份证的 18 位数字中，有关于出生年、月、日、性别等的信息，例如第 7～10 位、第 11～12 位、第 13～14 位分别表示出生年、月、日，第 17 位表示性别，奇数指男

性，偶数指女性。编程显示这 4 个信息。

求解方法：输入一个虚构的身份证号码(例如 510102196109089543)，利用整除和取余运算分别取出其中的出生年、月、日和性别信息。

源程序如下：

```
id=int(input("身份证号码:"))
year=id//10000000000%10000
month=id//1000000%100
day=id//10000%10
sex=id%100//10
print("出生日期:",year,"年",month,"月",day,"日")
if sex %2==1:
    print("此人男性")
else:
    print("此人女性")
```

本例中使用整除和取余运算分别取身份证号码中的出生年、月、日和性别信息。

运行结果如图 3-12 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
身份证号码: 510102196109089543
出生日期:  1961 年 9 月 8 日
此人女性
```

图 3-12　例 3-11 的运行结果

3. 位运算

位运算是以二进制整数的数位为基础进行运算的，没有进位与借位问题，也不涉及符号位，所以又称为按位运算。常用的按位运算符如表 3-6 所示。

表 3-6　按位运算符

运算符	名　称	描　　述
&	按位与	两个数进行运算，若对应两个二进制位均为 1，则结果为 1，否则为 0
|	按位或	两个数进行运算，若对应两个二进制位均为 0，则结果为 0，否则为 1
^	按位异或	两个数进行运算，若对应两个二进制位相异，则结果为 1，否则为 0
~	按位取反	对每个二进制位取反，即 1 变为 0，0 变为 1
<<	左移	全部二进制位左移指定位，高位丢弃，低位补 0
>>	右移	全部二进制位右移指定位，低位丢弃，高位沿用

说明：左移 n 位是将该值乘以 2^n，右移 n 位是将该值除以 2^n。

【例 3-12】　位运算示例。

源程序如下：

```
a=0o400                                        #八进制数
b=a<<1                                         #左移 1 位
print("a=",a,"\nb=",b)
```

本例中的第 1 行使用八进制常量 0o400,对应的十进制数值为 256;第 2 行使用左移运算<<,这里左移 1 位是将该值乘以 2,即最低位添 0,则得到 512。

运行结果如图 3-13 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
a= 256
b= 512
```

图 3-13　例 3-12 的运行结果

3.5.2　实型数据

实型(float)常量又称为浮点型常量,常常用于表示有小数部分的数据。尤其是对于一些很大或很小的数以及一些其他非整数的十进制数据,都必须使用浮点型常量。浮点数代表具有小数部分的十进制数,它们可以用自然计数法或科学计数法来表示。其中,自然计数法是由一个整数部分和一个纯小数部分组合来表示浮点型数据的,例如 123.456。科学计数法是使用指数幂形式来表示浮点型数据的,例如 12.3456e1、1.23456e2 和 0.123456e3。

由于只有相同或兼容类型的数据才能进行计算,所以当参与运算的数据中有一个为浮点数时,会自动将另一个数转换为浮点数,其结果也是浮点数。在 Python 中除法运算与数学保持一致。即两个整数相除 8/5 的结果为 1.6,不再是 1,也就是除法中的两个数据在 Python 中已在运算前进行过类型转换,然后相除并得到浮点数形式的商。另外,Python 使用符号"//"表示整除,又称为 floor 除法,例如 8//5 的结果为 1,6.5//2.5 的结果为 2.0。

有些数据类型是兼容的,可以在进行数据类型转换后进行处理,举例如下。

【例 3-13】 数据类型转换示例。

源程序如下:

```
ch1=65
ch2=66
print("ASCII 码 65 表示:\t%c"%ch1)
print("ASCII 码 66 表示:\t%c"%ch2)
N1=0x12
print("N1 转换为十进制:\t",N1)
N2=0x123
print("N2 转换为十进制:\t",N2)
N3=123456789
print("N3 转换为十进制:\t",N3)
```

本例中的%c 和%d 是描述字符和整数的格式符,运行结果如图 3-14 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
ASCII码65表示:   A
ASCII码66表示:   B
N1转换为十进制:    18
N2转换为十进制:    291
N3转换为十进制:    123456789
```

图 3-14　例 3-13 的运行结果

3.5.3　分数型数据

分数(fractions)数据由分子和分母这两个整数构成,分别作为构造函数 Fraction()中的两个参数,即 Fraction(x,y)表示分数 x/y。使用分数后,可避免浮点数的某些计算误差,但在使用分数前需要导入分数模块 fractions。由于 Python 通过重载技术实现分数的四则运算,所以分数的四则运算可以直接书写,系统会自动计算出对应的分数运算结果。

【例 3-14】 分数数据示例。

源程序如下:

```
#导入分数模块 fractions
from fractions import Fraction
a=Fraction(1,2)
b=Fraction(1,3)
print("a-b=",a-b)
print("a+b=",a+b)
print("a * b=",a * b)
print("a/b=",a/b)
```

运行结果如图 3-15 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
a-b= 1/6
a+b= 5/6
a*b= 1/6
a/b= 3/2
```

图 3-15　例 3-14 的运行结果

本例中的 Fraction(1,2)表示生成分数数据 1/2。注意,分数数据是不能由实型数据 0.5 替代的,因为数据类型不同。

3.5.4　复数型数据

正如数学运算那样,Python 中的复数是由实部和虚部构成的数据,例如 2+5J(其中 J 可以是小写字母 j),显然虚部是必须存在的。标记虚部的后缀符号可以是 j 或 J。由于 Python 通过重载技术实现复数的四则运算,所以复数的四则运算可以直接书写,系统会自动计算出对应的复数运算结果。

【例 3-15】 复数数据示例。

源程序如下:

```
a=2+3J
b=3+2J
print("a-b=",a-b)
print("a+b=",a+b)
print("a * b=",a * b)
print("a/b=",a/b)
```

运行结果如图 3-16 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
a-b= (-1+1j)
a+b= (5+5j)
a*b= 13j
a/b= (0.9230769230769231+0.38461538461538464j)
```

图 3-16 例 3-15 的运行结果

3.6 字符串型数据

Python 可以使用字符串(string)类型,并没有表示单个字符的数据类型,即单个字符将作为长度为 1 的字符串进行处理。另外,Python 中的内置数据类型 string 可用于实现字符串的表示和处理。

3.6.1 字符串常量

表示字符串常量,可以通过单引号或双引号界定,具体用法如表 3-7 所示。

表 3-7 字符串常量

字符串常量	说　明
单引号	包含在单引号中的字符串,其中可以内含双引号
双引号	包含在双引号中的字符串,其中可以内含单引号
三单引号	包含在三单引号中的字符串,可以表示跨行的字符串

【例 3-16】 字符串常量示例。

源程序如下:

```
print("************************")
print("*   Python Programming   *")
print("************************")
```

本例中定义 3 个字符串常量并分别进行显示。

运行结果如图 3-17 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
************************
*   Python Programming   *
************************
```

图 3-17 例 3-16 的运行结果

3.6.2 转义字符

Python 的字符型常量具有 16 位值,并可以转换成整数,可以用整数运算符对它进行操作。全部可见的 ASCII 字符(除控制字符)都是可以直接用引号界定的,例如"A"、"5"、"*"等,但是对于那些不能直接输入的字符即控制字符,则可以通过转义字符来表示,如表 3-8 所示。

表 3-8　转义字符

转义字符	说　　明
\(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空符号
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\ddd	八进制数 ddd 代表的字符,例如\012 代表换行
\xdd	十六进制数 dd 代表的字符,例如\x0a 代表换行
\other	其他的字符以普通格式输出

【例 3-17】　含转义字符的字符串示例。

源程序如下:

```
s="a\tb\tc"            #转义字符\t 表示横向制表符
d="\101"               #表示八进制数的 ASCII 码
e="\x41"               #表示十六进制数的 ASCII 码
print("s=",s)
print("d=",d)
print("e=",e)
```

运行结果如图 3-18 所示。



图 3-18　例 3-17 的运行结果

3.6.3　字符串测试函数

Python 中内置许多字符串测试函数,测试函数的返回结果是布尔类型,即结果是 True 或 False。测试函数均要求字符串至少包含一个字符,否则将返回 False。常用字符串测试函数如表 3-9 所示。

表 3-9　字符串测试函数

函　　数	说　　明
isalpha()	判断是否全部为字母,若是,则返回 True,否则返回 False
isalnum()	判断是否全部为字母和数字,若是,则返回 True,否则返回 False
isdigit()	判断是否全部为数字,若是,则返回 True,否则返回 False
islower()	判断是否全部为小写字母,若是,则返回 True,否则返回 False
isupper()	判断是否全部为大写字母,若是,则返回 True,否则返回 False
isspace()	判断是否全部为空格,若是,则返回 True,否则返回 False

【例 3-18】 字符串测试函数示例。

源程序如下:

```
str="Python 6.3.3"
dgt="123456789"
print("isalpha():\t",str.isalpha())
print("isalnum():\t",str.isalnum())
print("isdigit():\t",str.isdigit())
print("islower():\t",str.islower())
print("isupper():\t",str.isupper())
print("isspace():\t",str.isspace())
print("123456789:\t",dgt.isalnum())
```

运行结果如图 3-19 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
isalpha():      False
isalnum():      False
isdigit():      False
islower():      False
isupper():      False
isspace():      False
123456789:      True
```

图 3-19　例 3-18 的运行结果

3.6.4　字符串运算符

字符串运算符如表 3-10 所示。

表 3-10　字符串运算符

运算符	描　　述
+	字符串左右连接
*	重复表示字符串
[]	通过索引号获取字符串中的指定字符
[:]	截取字符串中的连续部分
In	成员运算符,若字符串中包含指定字符则返回 True,否则返回 False

续表

运算符	描　　述
not in	成员运算符，若字符串中不含指定字符，则返回 True，否则返回 False
r/R	原始字符串，字符串均按字面来使用，无转义特殊或不可显示字符

【例 3-19】 字符串运算示例。

源程序如下：

```
s1="Python"
s2="Programming"
s3=s1+s2
s4=s1 * 3
s5=s2[0:6]
s6="python" in s1
s7="python" not in s1
print("s1:\t",s1)
print("s2:\t",s2)
print("s3:\t",s3)
print("s4:\t",s4)
print("s5:\t",s5)
print("s6:\t",s6)
print("s7:\t",s7)
```

运行结果如图 3-20 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
s1:     Python
s2:     Programming
s3:     PythonProgramming
s4:     PythonPythonPython
s5:     Progra
s6:     False
s7:     True
```

图 3-20　例 3-19 的运行结果

3.6.5　字符串内置函数

字符串内置函数如表 3-11 所示。

表 3-11　字符串内置函数

函　　数	描　　述
capitalize()	串首字符大写
center(width)	返回原串居中部分并用空格填充至 width 的新串
count(str,beg=0,end=len(string))	返回子串 str 在原串 string 中出现的次数，若 beg 或 end 指定，则返回指定范围内子串 str 出现的次数
decode(encoding='UTF-8',errors='strict')	以 encoding 编码格式解码原串 string。除非 errors 指定 ignore 或 replace，否则若出错，则默认抛出 ValueError

续表

函　　数	描　　述
expandtabs(tabsize=8)	将原串中 Tab 符转换成空格，Tab 默认空格数为 8
find(str,beg=0,end=len(string))	检测子串 str 是否包含在原串 string 中。若指定 beg 和 end，则检查是否包含在指定范围内，若是则返回开始索引值，否则返回－1
format()	格式化字符串
index(str,beg=0,end=len(string))	与 find()一样，若子串 str 不在原串 string 中，则抛出异常
join(seq)	以原串作为分隔符，将子串 seq 中的所有元素合并为新串
ljust(width)	返回原串，左对齐并用空格填充至长度为 width 的新串
lower()	转换原串中所有大写字母为小写
lstrip()	删除原串左侧的空格
max(str)	返回 str 串中的最大字母
min(str)	返回 str 串中的最小字母
replace(str1,str2,num=count(str1))	将原串中的子串 str1 替换成子串 str2，若指定 num 则替换不超过 num 次
rfind(str,beg=0,end=len(string))	与 find()类似，不过是从右边开始
rindex(str,beg=0,end=len(string))	与 index()类似，不过是从右边开始
rjust(width)	返回长度的 width 的串，原串右对齐，前面用空格填充
rpartition(str)	与 partition()类似，不过是从右边开始查找
rstrip()	删除原串末尾的空格
split(str="",num=count(str))	以子串 str 为分隔符将原串切片，若指定 num 则仅分隔 num 个子串
strip([obj])	在原串上调用 lstrip()和 rstrip()
swapcase()	翻转原串中的大小写字母
title()	返回"标题化"的原串，即单词以大写开始，其余字母小写
translate(str,del="")	根据子串 str 给出的表转换原串的字符
upper()	转换原串中的小写字母为大写形式
zfill(width)	返回长度为 width 的新串，原串右对齐，前面填充 0

【例 3-20】 字符串内置函数示例。

源程序如下：

```
#初始化字符串
str="Python Pragramming"
#将字母转换成大写形式
s1=str.upper()
#将字母转换成小写形式
s2=str.lower()
print("原始字符串:\t",str)
```

```
print("转换成大写:\t",s1)
print("转换成小写:\t",s2)
```

运行结果如图 3-21 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
原始字符串:        Python Pragramming
转换成大写:        PYTHON PRAGRAMMING
转换成小写:        python pragramming
```

图 3-21　例 3-20 的运行结果

字符串是任何信息处理的基础，在第 4 章中将详细介绍字符串及其编程。

3.7　布尔型数据

布尔型(boolean)数据又称为逻辑型数据，用于表示条件判断。实际上，数学中的关系运算也可以进行条件判断，这是布尔型数据中的较简单形式。下面首先说明关系运算，然后再介绍布尔型数据和运算。

George Boole(1815—1864)是英国杰出的数学家，他最早开创了逻辑代数的理论。后来，人们为了纪念他，就将逻辑类型数据称为布尔类型数据。

3.7.1　关系运算

关系运算符用于比较运算，只有相同数据类型的两个表达式才能进行比较，运算结果只能为两种逻辑值 True 或 False。关系运算符的内容如表 3-12 所示。

表 3-12　关系运算符

名称	关系运算符	示例	名称	关系运算符	示例
大于	＞	20＞10	不等于	!＝	20!＝10
小于	＜	20＜10	大于等于	＞＝	20＞＝10
等于	＝＝	20＝＝10	小于等于	＜＝	20＜＝10

例如：关系表达式 50＞100 的结果为 False。

说明：

(1) 关系运算符确定一个操作数与另一个操作数之间的关系，运算的结果为逻辑类型，它经常用于条件语句和循环语句中的条件判断。

(2) 英文符号比较大小是按其 ASCII 码的码值大小进行比较的。汉字字符串比较大小使用相应的汉语拼音字母代替该汉字。然后，再按英文字母比较大小的规则进行比较。

关系比较运算符与数学中的比较运算一致，一般是两个类型相同的对象进行运算，一般而言不同对象之间不能进行比较。数字进行比较的是数值大小，字符串进行比较的是对应 ASCII 码的码值大小，所以可以用＜、＜＝、＞、＞＝、＝＝和!＝这些运算符来连接两个字符串，依次比较字符串中各个字符的 ASCII 值，得到比较的逻辑结果为 True 还是 False。

列表和元组也是可以比较的，操作逻辑与字符串类似。

3.7.2 布尔常量

布尔型数据只有两个：True 和 False，分别表示真或假。在 Python 中，空数据类型的布尔运算结果均为 False。实际上，布尔型数据是整型数据的子类。条件比较运算的结果就是布尔型数据，因此布尔型数据通常作为程序中的分支选择或循环重复的测试条件来使用。

说明：

(1) 所有关系表达式的返回值都是布尔型常量 True 或 False。

(2) 在输出逻辑类型的数据时，结果只能是 True 或 False。

(3) 布尔型数据不能转换成另外的数据类型。

3.7.3 布尔运算

布尔运算符是对一个或两个逻辑型表达式实施逻辑运算，只能产生逻辑型的运算结果，即 True 或 False。在 Python 中，只有逻辑与(and)、逻辑或(or)和逻辑非(not)3 个逻辑运算符。逻辑运算符的含义如表 3-13 所示。

表 3-13 逻辑运算符

名称	逻辑运算符	示例	说　　明
逻辑与	and	*a* and *b*	*a* 和 *b* 同时为 True 则结果为 True，其余都为 False
逻辑或	or	*a* or *b*	*a* 和 *b* 同时为 False 则结果为 False，其余都为 True
逻辑非	not	not *a*	*a* 为 True 则结果为 False，*a* 为 False 则结果为 True

注意：在运用逻辑与(and)和逻辑或(or)运算符时，可以通过合理地安排表达式的执行顺序来提高程序的运行效率。例如，对于逻辑与(and)运算符，如果左侧的表达式结果为 False，就不必再计算右侧表达式了。同样，对于逻辑或(or)运算符，如果左侧的表达式结果为 True，也就不必计算右侧表达式了。这种运算处理类似于零乘任何数为零，也没有相乘。

【例 3-21】 编程判断 2017 年是否为闰年。(提示：闰年条件是能被 4 整除但不能被 100 整除，或能被 400 整除。)

是否闰年的布尔型表达式为 year%4==0 and year%100!=0 or year%400==0。

源程序如下：

```
year=int(input("今年:"))
flag=year%4==0 and year%100!=0 or year%400==0;
if flag:
    print(year,"年是闰年")
else:
    print(year,"年不是闰年")
```

本例中的第 1 行要求键盘输入年份(实际输入为 2017)，第 2 行直接使用布尔型表达式来表示闰年条件，第 3～6 行使用 if 语句得到判断结果并显示。

运行结果如图 3-22 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
今年: 2017
2017 年不是闰年
```

图 3-22　例 3-21 的运行结果

3.8　序列数据

序列对象中的元素是有序存放的,也就是说序列元素都有指定的位置索引,访问序列元素也是通过位置索引进行的。实际上,有序存储和按位置索引的序列对象共有许多通用的特点和操作。在 Python 中,全部序列数据均可以进行的操作包括索引访问、切片、加、乘、检查成员、求序列长度、求最大值、求最小值等。

下面分别介绍的列表、元组、字典和集合均是序列数据。

3.8.1　列表

1. 列表及其操作

列表(list)是序列对象中的一种,属于可变的序列类型。列表的元素用方括号界定,元素之间由逗号分隔。列表还是一个容器类型的对象,也就是说,列表的元素可以是任何类型的对象,例如数字、字符串、列表、另一元组、字典等都可以作为列表中的元素,并且元素个数没有限制。

列表的元素是按从左到右顺序存放的,元素的位置索引是从 0 开始索引的整数。通过这个位置索引来访问列表元素,位置索引要放置在方括号内,一般引用形式如下:

```
列表名[索引]
```

列表元素可以是由任何类型的混合数据构成。无论多复杂的数据结构,都是按照位置索引的。不包含任何元素的列表为空列表,表示为[]。还需要说明的是,列表是一种可变对象类型,也就是说可以修改列表。一方面,可以通过索引位置修改原来列表的一个或多个元素的值,或者说为列表的索引位置赋值时,均不会生成新的列表;另一方面,列表中的元素可以任意增加,例如向列表中添加元素的操作如 append、extend 均可以增加列表长度。

列表是 Python 中最基本的数据结构,也是最常用的数据类型。列表的数据项不需要具有相同的类型。列表中的每个元素都分配有一个索引数字来表示相应的位置,第 1 个索引是 0,第 2 个索引是 1,依此类推。也可以使用负数索引来表示列表元素,最右(后)一个元素的索引是 −1,倒数第 2 个索引是 −2,以此类推,如图 3-23 所示。

从左到右索引	0	1	2	3	4	5	
	P	y	t	h	o	n	
	-6	-5	-4	-3	-2	-1	从右到左索引

图 3-23　索引示意图

(1) 创建列表。要创建一个列表,只需要将全部数据项(可以是不同的数据类型)用逗号分隔,并用方括号括起来即可。

【例 3-22】 创建列表示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>my_list=[1, 2, 3, 4, 5]                #创建列表并初始化
>>>my_list                                #显示列表
[1, 2, 3, 4, 5]
>>>my_list =["a", "b", "c", "d"]          #第 2 次初始化列表
>>>my_list                                #显示列表
['a', 'b', 'c', 'd']
```

与字符串的索引一样,列表索引也是从 0 开始的,且可以进行切片、组合等操作。

(2) 访问列表元素。上面使用列表名能够访问整个列表元素,但若只访问列表中的一个或部分元素,则需要使用索引来指定列表元素。当然,使用方括号形式也可以截取部分元素。

【例 3-23】 访问列表元素示例。

```
>>>my_list =[1, 2, 3, 4, 5]               #创建列表并初始化
>>>my_list [4]=8"                         #修改列表中的第 5 个元素, 即用 8 替代 5
>>>my_list                                #显示列表
[1, 2, 3, 4, 8]
```

(3) 附加列表元素。除使用索引对列表中的元素进行修改或更新外,Python 还提供 append()内置函数在列表末尾附加列表元素。

【例 3-24】 附加列表元素示例。

源程序如下:

```
lst=[12,23,34,45]                         #创建列表并初始化
print("原始列表:",lst)
lst.append(56789)                         #在列表末尾附加一个元素
print("更新列表:",lst)
```

运行结果如图 3-24 所示。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
原始列表: [12, 23, 34, 45]
更新列表: [12, 23, 34, 45, 56789]
```

图 3-24 例 3-24 的运行结果

(4) 删除列表元素。要删除列表元素则可以使用 del 语句。

【例 3-25】 删除列表元素示例。

源程序如下:

```
lst=[12,23,34,45,56]
print("原始列表:",lst)
del lst[1]
print("更新列表:",lst)
```

运行结果如图 3-25 所示。

除以上列表操作外,Python 还提供许多内置函数支持列表操作,下面进行介绍。

```
====================== RESTART: D:/Python36/ch3/p1.py ======================
原始列表: [12, 23, 34, 45, 56]
更新列表: [12, 34, 45, 56]
```

图 3-25　例 3-25 的运行结果

2. 列表函数

列表操作函数分为两类，一类是对列表进行整体操作，另一类只是对列表中的元素进行操作。

对列表进行整体操作的函数，如表 3-14 所示。

表 3-14　对列表进行整体操作

函　　数	说　　明
cmp(＜list1＞,＜list2＞)	比较两个列表中的元素大小
len(＜list＞)	计算列表中的元素个数
max(＜list＞)	计算并返回列表元素中的最大值
min(＜list＞)	计算并返回列表元素中的最小值
list(seq)	将元组转换为列表

对列表中的元素进行操作的函数，如表 3-15 所示。

表 3-15　对列表中的元素进行操作

方　　法	说　　明
append(＜obj＞)	在列表末尾添加新的对象
count(＜obj＞)	统计某个元素在列表中出现的次数
extend(＜seq＞)	在列表末尾一次性追加另一个序列中的多个值
index(＜obj＞)	从列表中找出某个值第一个匹配项的索引位置
insert(＜index＞,＜obj＞)	将指定对象插入列表
pop(＜obj＞=list[-1])	移除列表中的一个元素(默认为最后元素)并返回该元素的值
remove(＜obj＞)	移除列表中某个值的第一个匹配项
reverse()	倒置列表中元素
sort([＜func＞])	将列表元素从小到大排序

3. 程序示例

【例 3-26】　用列表分别表示 2016 年和 2017 年中的每月天数，并调用函数找出 12 个月中的最少天数，以及每月天数的升序排列。

源程序如下：

```
lst=[31,28,31,30,31,30,31,31,30,31,30,31]
print("2017年每月天数:")
for i in range(0,12,1):
      print(lst[i],end="  ")
```

```
lst[1]=29
print()
print("2016年每月天数:")
for i in range(0,12,1):
      print(lst[i],end="  ")
print()
print("2016年最少天数:",min(lst))
print()
lst.sort()
print("2016年天数升序:")
for i in range(0,12,1):
      print(lst[i],end="  ")
```

本例中通过调用 min()函数找出 12 个月份中的最少天数,其后调用 sort()函数实现列表元素的升序排列。另外,参数 end=" "表示不分行,print()语句表示换行。

运行结果如图 3-26 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
2017年每月天数:
31  28  31  30  31  30  31  31  30  31  30  31
2016年每月天数:
31  29  31  30  31  30  31  31  30  31  30  31
2016年最少天数:  29

2016年天数升序:
29  30  30  30  30  31  31  31  31  31  31  31
```

图 3-26　例 3-26 的运行结果

【例 3-27】 求出平面上两个坐标点之间的距离。

下面比较用变量实现和用列表实现的两种处理方法。

(1) 用变量实现。

源程序如下:

```
import math
x1=1
y1=2
x2=3
y2=4
dis=math.sqrt((x2-x1) * (x2-x1)+(y2-y1) * (y2-y1))
print(dis)
```

运行结果:

```
2.8284271247461903
```

(2) 用列表实现。

源程序如下:

```
import math
xy=[1,2,3,4]
dis=(xy[2]-xy[0]) * (xy[2]-xy[0])+(xy[3]-xy[1]) * (xy[3]-xy[1])
dis=math.sqrt(dis)
```

```
print(dis)
```

从以上两个程序中可以发现，使用列表能够更容易表示大量数据，这是简单变量无法实现的。

3.8.2 元组

元组(tuple)也是一种序列类型，这是一种不可变的序列对象。在没有歧义的情况下，元组也可以没有圆括号。所以，若元组中只有一个元素，不加括号时也要有逗号，否则就无法和单个数据进行区分了。

元组操作和列表操作的非常类似，也是按照位置索引来访问元素的，也有加、乘、索引、切片等操作。不同之处在于元组为不可变类型数据，即不能修改其中的任何元素，而列表是可以修改的。所以，凡是可以修改序列内容的操作，而在元组中都是没有的，这样就使元组的应用范围比列表小很多。尽管元组不能修改，但可以整体重新赋值，这样将生成一个新的元组。

【例 3-28】 元组运算示例。

源程序如下：

```
#创建元组并初始化为 10 个元素
t=(1,2,3,4,5,6,7,8,9,10)
print("len(t)=",len(t))              #求出元组中的元素个数
print("sum(t)=",sum(t))              #求出元组中的元素之和
print("max(t)=",max(t))              #求出元组中的最大元素值
print("min(t)=",min(t))              #求出元组中的最小元素值
```

运行结果如图 3-27 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
len(t)= 10
sum(t)= 55
max(t)= 10
min(t)= 1
```

图 3-27 例 3-28 的运行结果

3.8.3 字典

字典(dict)也是一种可变的类型，其中可以存储任何类型的数据。

字典由许多键值组成，每个键值(<key>：<value>)对用冒号分隔，每个键值对之间用逗号分隔，整个字典包括在一对花括号中，一般使用格式如下：

```
d ={<key1>:<value1>, <key2>:<value2>, … }
```

说明：

(1) 这里的<key>值必须是唯一的，但<value>值则可以重复书写。

(2) <value>值可以是任何数据类型，但<key>值必须是不可变的数据。

注意：

(1) 不允许一个键在字典中出现两次，若出现两次则系统将显示出错信息。

(2) <key>值必须是不可变数据，所以只能用数字、字符串或元组表示。

1. 创建字典并访问值

【例 3-29】 创建字典并访问值示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>my_dict={"Lenovo":3200,"ThinkPad":3600,"Dell":3400}
>>>my_dict
{'Lenovo': 3200, 'ThinkPad': 3600, 'Dell': 3400}
>>>my_dict["Lenovo"]                   #访问字典中的值
3200
```

2. 修改字典值

【例 3-30】 修改字典值示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>my_dict={"Lenovo":3200,"ThinkPad":3600,"Dell":3400}
>>>my_dict["Lenovo"]=4000              #用 4000 替代原来的 3200
>>>my_dict["Lenovo"]
4000
```

3. 删除字典元素

删除一个字典元素可以使用 del 语句,一般引用格式如下:

```
del D.[<key>]
```

功能是将<key>及其对应的<value>从指定字典 D 中删除,若 del 语句后没有参数则将删除整个字典。

【例 3-31】 删除字典元素示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>my_dict={"Lenovo":3200,"ThinkPad":3600,"Dell":3400}
>>>del my_dict["Lenovo"]               #删除一个字典元素
>>>my_dict
{'ThinkPad': 3600, 'Dell': 3400}
```

【例 3-32】 删除字典示例。

源程序如下:

```
my_dict={"Lenovo":3200,"ThinkPad":3600,"Dell":3400}
print("第 1 次显示字典:\n\t",my_dict)
del my_dict
print("第 2 次显示字典:\n\t",my_dict)
```

本例中使用没有参数的 del 语句将删除字典,其后试图显示字典时系统会产生 NameError 异常。

运行结果如图 3-28 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
第1次显示字典:
         {'Lenovo': 3200, 'ThinkPad': 3600, 'Dell': 3400}
Traceback (most recent call last):
  File "D:/Python36/ch3/p1.py", line 4, in <module>
    print("第2次显示字典: \n\t",my_dict)
NameError: name 'my_dict' is not defined
```

图 3-28　例 3-31 的运行结果

3.8.4　集合

1. 集合类型及其定义

集合(set)是一种与列表和字典完全不同的序列数据,特点就是集合对象中不允许有重复的元素。集合有不同的两种类型:可变集合和不可变集合,其中可变集合可以添加和删除元素,而不可变集合则不允许修改。在 Python 中,创建可变集合可以调用构造函数 set(),并由序列或其他可迭代对象数据来表示。创建不可变集合则可以调用构造函数 frozenset()。在 Python 3.6 中,可以直接将所有元素写在一对花括号中来定义集合。

关于集合的主要运算有成员检测和集合操作。除了成员检测运算符 in 外,还有数学意义的交集、并集、差集、子集等运算,集合还可以调用函数 len()来获取集合元素的个数。

集合运算可以直接利用集合运算符,还可以调用集合对象操作的函数。

2. 集合运算符

集合运算符用于连接两个集合并进行计算,如表 3-16 所示。

表 3-16　集合运算符

运算符	名称	说　　明
set_a&set_b	求交集	找出 set_a 和 set_b 中共同元素
set_a|set_b	求并集	将 set_a 和 set_b 中的全部元素合并起来
set_a-set_b	求差集	将 set_a 中为 set_b 的元素从集合中删除
set_a^set_b	异或	找出 set_a 和 set_b 中独有的元素

3. 集合操作函数

集合对象内置的运算方法返回结果是集合对象,如表 3-17 所示。

表 3-17　集合操作函数

函　　数	说　　明
set_a.intersection(set_b)	返回 set_a 和 set_b 的交集
set_a. union (set_b)	返回 set_a 和 set_b 的并集
set_a. difference (set_b)	返回 set_a 和 set_b 的差集
set_a.issubset (set_b)	返回 set_a 是否是 set_b 的子集的逻辑值
set_a.issuperset (set_b)	返回 set_a 是否是 set_b 的父集的逻辑值

【例 3-33】 创建集合及其成员测试。

源程序如下:

```
#创建集合并初始化
my_color={"blue","red","blue","green","red","black"}
print(my_color)                     #集合中仅含 4 个元素
print("red"in my_color)             #成员测试为 True
print("gray"in my_color)            #成员测试为 False
```

运行结果如图 3-29 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
{'black', 'green', 'red', 'blue'}
True
False
```

图 3-29　例 3-33 的运行结果

【例 3-34】 集合运算示例。

源程序如下：

```
a=set("0123456789101112")           #使用构造函数创建集合 a 并初始化
b=set("13579")                      #使用构造函数创建集合 b 并初始化
print("集合 a: \t",a)
print("集合 b: \t",b)
print("集合 a-b: ",a-b)             #求差集运算
print("集合 a|b: ",a|b)             #求并集运算
print("集合 a&b: ",a&b)             #求交集运算
print("集合 a^b: ",a^b)             #异或运算
```

运行结果如图 3-30 所示。

```
======================= RESTART: D:/Python36/ch3/p1.py =======================
集合a:    {'3', '7', '5', '0', '2', '1', '9', '8', '6', '4'}
集合b:    {'3', '7', '5', '1', '9'}
集合a-b:  {'0', '2', '8', '6', '4'}
集合a|b:  {'3', '7', '5', '0', '2', '1', '9', '8', '6', '4'}
集合a&b:  {'3', '7', '5', '1', '9'}
集合a^b:  {'0', '2', '8', '6', '4'}
```

图 3-30　例 3-34 的运行结果

习　题　3

一、简答题

1. 什么是编码风格？Python 语言的编码风格有哪些？
2. 什么是简单程序？它严格遵循的处理时序是什么？
3. 简述组合符号、标识符、关键字和预定义标识符的定义与区别。
4. Python 遵循的标识符命名规则有哪些？
5. 什么是基本数据类型？什么是复合数据类型？
6. 什么是不可变数据类型？什么是可变数据类型？
7. 什么是常量？什么是变量？
8. Python 语言有哪些数字类型？数字数据能进行的计算有哪些？
9. 什么是转义字符？简述如何使用转义字符。

10. Python 语言中有哪些字符串运算符？

11. 什么是布尔型？Python 语言中有哪些布尔运算符？

12. 什么是序列数据？Python 语言中有哪些序列数据？

13. 什么是列表？Python 语言中有哪些列表运算符？

14. 什么是元组？Python 语言中有哪些元组运算符？

15. 什么是字典？Python 语言中有哪些字典运算符？

16. 什么是集合？Python 语言中有哪些集合运算符？

17. 设定三角形的三条边分别为 a、b、c 且 $a \leqslant b \leqslant c$，写出测试下面 3 种三角形的布尔表达式：

(1) 直角三角形；

(2) 等边三角形；

(3) 等腰三角形。

二、编程题

1. 生成列表[1,4,9,16,25,36,49]。

2. 生成字典["one":1,"two":4,"three":9,"four":16,"five":25,"six":36,"seven":49]。

3. 编程：将 7 个英文单词 monday，tuesday，wednesday，thursday，friday，saturday，sunday 放入列表 week[]中，并删除最后 2 个单词又放入列表 weekday[]中。

4. 分别初始化两个长度为 20 的空格串和星号串，使用切片操作输出如下形式的平行四边形。

```
      ********************
     ********************
    ********************
   ********************
  ********************
 ********************
********************
```

5. 输出如下形式的等腰三角形。

```
      *
     ***
    *****
   *******
  *********
 ***********
*************
```

6. 编程：从键盘输入立方体的长、宽和高，计算体积。

7. 编程：从键盘输入三角形的 3 条边的长度并计算面积。

(提示：使用海伦公式 $S=\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}$，其中 $p=(a+b+c)/2$。)

三、操作题

把列表当作堆栈使用。堆栈(stack)是一种采用后进先出(LIFO)策略的数据结构，就是现实生活中火车进站策略。先进站的火车后开车，后进站的火车先开车。列表可以实现堆栈操作，出栈操作是删除列表中的第 1 个元素，入栈操作是在列表的第 1 个元素前插入。

在 IDLE 交互环境中，通过操作实现如下要求。

（1）初始化列表：

```
["monday", "tuesday", "thursday", "wednesday", "friday", "saturday", "sunday"]
```

（2）出栈操作得到列表：

```
["thursday", "wednesday", "friday", "saturday", "sunday"]
```

（3）入栈操作得到列表：

```
["星期一", "星期二", "thursday", "wednesday", "friday", "saturday", "sunday"]
```

操作完成后，要求将操作过程与运行结果截图并存入"习题 3.doc"文档中。

第 4 章　流 程 控 制

在程序设计过程中，程序员经常会使用各种各样的控制语句，以便根据问题求解的需要来控制程序执行的流程。Python 与其他计算机语言一样，全部控制语句也可以分为 3 种类型：分支控制语句、循环控制语句和跳转控制语句。其中，分支控制语句是根据表达式或变量的取值来选择所执行的分支；循环控制语句是在一定条件下使程序能反复执行一个程序段；跳转控制语句允许程序在某种条件下改变常规执行顺序，直接跳转到某个特定的位置继续执行。要注意的是，这 3 种控制语句都将使程序不再按语句的书写顺序逐条执行。另外，后续章节中还将介绍异常处理、函数定义与调用、递归函数、模块调用等也都可以控制程序的执行流程。

本章将介绍实现分支选择和循环控制的语句及其编程，例如 if、while、for、continue、break、pass 语句和 range()函数；最后介绍的编程案例，包括列表处理、查找、排序和字符串处理。

4.1　简单程序与流程控制

简单程序的运行流程是完全固定的，程序的功能必将受到极大地限制。而程序中通过流程控制将扩大程序的描述能力。

4.1.1　简单程序

简单程序是指该程序自始至终按照语句序列的排列顺序，从头到尾逐条执行，它是程序设计中的顺序程序。

【例 4-1】　编程计算 5!。

源程序如下：

```
fact=1
fact=fact*1
fact=fact*2
fact=fact*3
fact=fact*4
fact=fact*5
print("fact=",fact);
```

运行结果如下：

```
fact=120
```

在运行本例中的程序时，计算机系统会逐条顺序执行全部语句，不过这种程序具有明显的缺点，若计算 100! 则需要写 100 条语句，所以使用 Python 提供的循环语句就能够有效

实现此类问题的求解。另外，许多程序也可以使用分支选择语句。

4.1.2 流程控制语句

要改变简单程序的执行流程，就可以使用控制语句。控制语句及其分类情况，如表 4-1 所示。

表 4-1 控制语句及其分类

名称	语句	功能
分支选择	if … else …	条件选择
循环控制	while … else …	当型(条件型)循环
	for … else …	计数型循环
辅助控制	break	终止循环
	continue	结束本次循环
	pass	空语句
	return	从函数中返回
	try … catch	异常处理

4.1.3 测试条件

选择控制和循环重复都是基于测试条件的结果才能实现的。

分支选择和循环控制的测试条件需要根据需求的不同而改变，常见的测试条件可以由关系运算、布尔运算、测试运算等构成。关系比较运算、布尔运算和测试运算的结果只有 True 和 False 两种取值。当条件表达式为 True 时，称为满足条件，否则称为不满足条件。例如，满足条件时将使 if 语句块内的指定分支得以执行，否则程序继续向下执行语句块外的语句。当然 Python 中的真假不仅仅限于比较运算和布尔运算的结果，任意的非零数据和非空数据结构也被视为 True，数字 0 和为空的数据均被视为 False。条件测试中为假的具体情况主要包括如下 5 种：

(1) 常量 None，表示没有数据；

(2) 数字类型的 0 值，例如 0 和 0.0；

(3) 全部空序列，例如空字符串""，空列表[]，空元组()；

(4) 空映射，即空字典{}；

(5) 自定义类中的方法__nonzero__()或__len__()，类实例化时均返回 0。

4.2 分支选择

简单程序是指程序严格按照语句的书写顺序逐条执行，这里是没有任何分支选择或循环重复操作的。然而，绝大多数的计算机处理并非如此“简单”，有时候需要根据条件进行分支选择，有时候需要根据条件控制循环过程。而计算机比其他计算工具的优越之处就在于

具有逻辑判断能力，这也是计算机能够实现分支选择和循环重复的基础。Python 引入逻辑判断后，程序不仅可以严格按照语句的书写顺序逐条执行，还能控制从一条语句跳到另一条语句，从一段程序跳到另一段程序。

在分支选择方面，Python 只支持 if 语句，若读者已经有一定的 C 语言编程经验，一定会发现 if 语句的重要性，它们可以按照不同的情况分别运行相应的语句块。下面将分别进行介绍。

if 是条件判断语句，也是复合型语句，在 if 中可以包括其他语句。复合语句在 Python 中有格式要求，一般首行语句以冒号结尾，下面被包含的语句（或语句块）向右缩进（本书使用缩进 4 格）。复合语句可以嵌套其他复合语句，因此缩进的层次也就对应嵌套的层次。

在其他高级语言中，一般用若干层括号来反映嵌套的层次关系，但是 Python 不依靠括号来体现嵌套层次，这一理念是 Python 自行独创的。在交互环境下，输入一条以冒号结尾的语句后，系统将自动向右缩进 4 格，从而使后续的语句成为被嵌套的下一个层次。这样使 Python 程序排列整齐和代码规范，也正是这种规范的缩进规则使得 Python 程序结构清晰直观，易于阅读。

利用 if 语句可以实现分支选择，具体形式分为单分支选择、双分支选择和多分支选择 3 种，下面分别进行介绍。

4.2.1 单分支选择

单分支选择的语句格式如下：

```
if <条件>:
    <语句块>
```

功能：如果测试＜条件＞的值为 True 时，则执行＜语句块＞，然后再执行后续的语句，否则就直接执行后续的语句。

【例 4-2】 找出 3 个整数 *a*、*b*、*c* 中的最大数。

源程序如下：

```
a=int(input("输入第 1 个数: "))
b=int(input("输入第 2 个数: "))
c=int(input("输入第 3 个数: "))
most=a                                  #设置最大数变量的初始值
if b>most: most=b                       #使 most 保存 a、b 中的较大数
if c>most: most=c                       #使 most 保存 a、b、c 中的最大数
print("最大数: ",most)
```

本例中的第 1～3 行声明变量 a、b、c 并接收输入的数据，其中的 input()函数只能接收字符串，所以由 int()函数将其转换成整数；第 4 行设置最大数变量 most 的初始值为变量 a 的值，第 5、6 行使用单分支选择根据条件取值情况修改变量 most 的值，第 7 行显示 3 个数中的最大值。

运行结果如图 4-1 所示。

注意：通常人们会使用 max 来表示最大值，但 Python 语言将 max 作为预定义标识符，

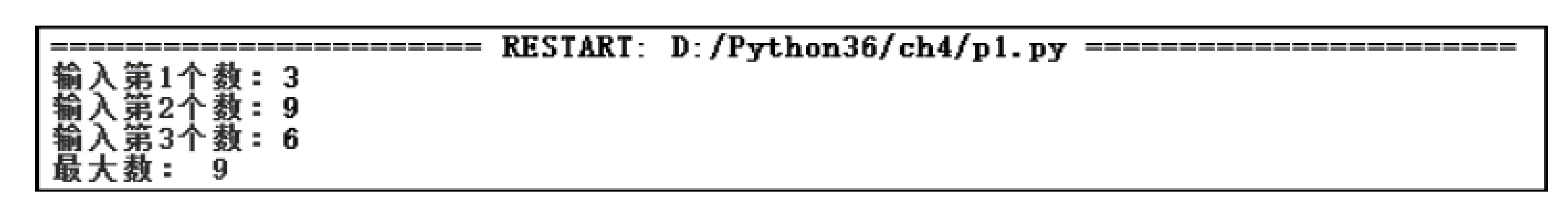


图 4-1　例 4-2 的运行结果

所以本书将不会将预定义标识符作为变量名。

4.2.2　双分支选择

双分支选择的语句格式如下：

```
if <条件>:
      <语句块 1>
else:
      <语句块 2>
```

功能：如果＜条件＞的值为 True 时，则执行＜语句块 1＞并结束该 if 语句，否则执行＜语句块 2＞中的语句并结束该 if 语句。

【例 4-3】　输入两个整数 x 和 y，按由小到大的顺序输出。

源程序如下：

```
x=int(input("x="))
y=int(input("y="))
if x>y:
    print("从小到大顺序：",y,x)
else:
    print("从小到大顺序：",x,y)
```

本例中的第 1、2 行要求输入变量 x 和 y 的值(本题中分别为 4 和 5)，第 3～6 行使用双分支选择语句输出正确结果。

运行结果如图 4-2 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
x=4
y=5
从小到大顺序： 4 5
```

图 4-2　例 4-3 的运行结果

【例 4-4】　输入 4 个整数，要求按由小到大的顺序输出。

在不用循环程序实现的前提下，只能按照排列组合方式逐个进行比较。

源程序如下：

```
a=int(input("第 1 个整数："))
b=int(input("第 2 个整数："))
c=int(input("第 3 个整数："))
d=int(input("第 4 个整数："))
if a>b : t=a;a=b;b=t
if a>c : t=a;a=c;c=t
if a>d : t=a;a=d;d=t
```

```
if b>c : t=b;b=c;c=t
if b>d : t=b;b=d;d=t
if c>d : t=c;c=d;d=t
print("排序结果：",a,b,c,d)
```

本例中通过 6 次比较并交换操作后，使 4 个整数按由小到大的顺序排列。若在程序运行时输入 6、8、1、4，则运行结果如图 4-3 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
第1个整数：6
第2个整数：8
第3个整数：1
第4个整数：4
排序结果： 1 4 6 8
```

图 4-3 例 4-4 的运行结果

说明：本例中的第 5 行用分号分隔 3 条语句，这不是 Python 推崇的编程风格。若按 Python 编程风格，则需要改写该语句如下：

```
if a>b :
    t=a
    a=b
    b=t
```

本书有时需要节省源程序的行数，只好合行书写。

4.2.3 多分支选择

Python 允许 if 语句进行嵌套，换句话说，在 if 语句的分支模块中，还可以包含其他的 if 语句，这样就可以构成多分支选择。多分支选择的语句格式如下：

```
if <条件 1>:
      <语句块 1>
elif <条件 2>:
      <语句块 2>
elif <条件 3>:
      <语句块 3>
      …
elif <条件 n>:
      <语句块 n>
else:
      <语句块 n+1>
```

说明：

＜条件 1＞、＜条件 2＞、…、＜条件 n＞：由布尔表达式或关系表达式作为条件。

＜语句块 1＞、＜语句块 2＞、…、＜语句块 $n+1$＞：指定不同分支上的语句块。

多分支选择语句的执行功能如图 4-4 所示。

【例 4-5】 判断字符属于阿拉伯数字、大写字母、小写字母或其他字符。

源程序如下：

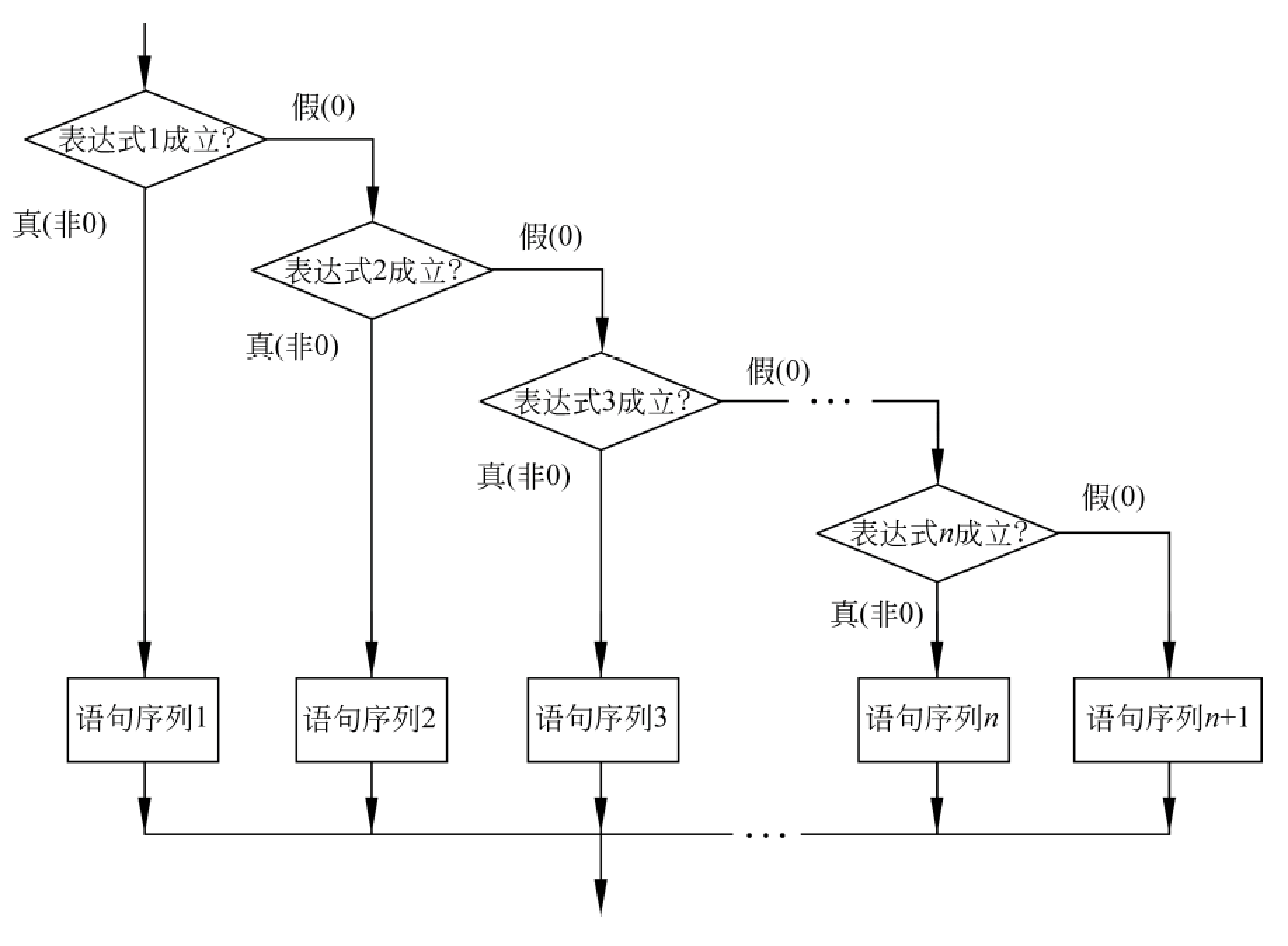


图 4-4　多分支选择语句

```
ch=input("ch=")                          #输入一个字符
if ch>="0" and ch<="9":
    print("这是阿拉伯数字:",ch)
elif ch>="A" and ch<="Z":
    print("这是大写字母:",ch)
elif (ch>="a" and ch<="z"):
    print("这是小写字母:",ch)
else:
    print("这是其他字符:",ch)
```

本例中的第 1 行要求输入一个字符,第 2～9 行使用多分支判断语句输出合理的提示信息。

若在程序运行时输入 H,则运行结果如图 4-5 所示。


```
====================== RESTART: D:/Python36/ch4/p1.py ======================
ch=H
这是大写字母: H
```

图 4-5　例 4-5 的运行结果

【例 4-6】　找出 3 个整数中的最大数。

源程序如下:

```
a=int(input("a="))
b=int(input("b="))
c=int(input("c="))
if a>b:
    if a>c:
```

```
        print("最大数: ",a)
    else:
        print("最大数: ",c)
else:
    if b>c:
        print("最大数: ",b)
    else:
        print("最大数 4: ",c)
```

本例中的语句块中嵌套的是双分支选择语句。

运行结果如图 4-6 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
a=4
b=2
c=9
最大数:  9
```

图 4-6　例 4-6 的运行结果

说明：与同样找出 3 个整数中的最大数的例 4-2 相比较，两个程序的求解算法完全不同。那么哪个算法好些？这必然会涉及评价标准，通常在保证程序正确的前提下，第二个标准就是可读性。而要提升可读性，遵循良好的编程风格至关重要。就语句书写而言，应该沿用正确的语句模式。而就编程而言，应该按照计算思维方式来求解问题。读者请自行根据可读性要求判断上面两个程序的优劣。

4.3　循环控制

在前面编写的程序，无论是简单程序还是分支程序，程序中的语句只执行一次或者完全不执行。这两类程序的共同特点，就是程序中的任意一条语句至多执行一次。然而在处理许多实际问题时，都必须使用循环控制方式，例如要求将一批数据求总和。

Python 支持 while 循环和 for 循环这两种循环控制。利用这两种循环流程，可以指定在一定条件下使程序能够重复执行某个语句块(即循环体)，从而完成所需的循环要求。实际上，任何复杂的计算任务都是可以由这两种循环控制语句来处理的。

下面分别介绍 while 循环和 for 循环。

4.3.1　while 语句

while 语句的一般格式如下：

```
while <条件>:
      <语句块 1>
[else:
      <语句块 2>]
```

功能：进入 while 循环将首先判断<条件>表达式，如果满足条件，就执行<语句块 1>，并重复这个过程直至<条件>不满足时为止。如果循环开始时<条件>就不满足，则循环体将不被执行，因此 while 循环体有可能一次也不被执行。而 else 部分是可选的，<语句块 2>

是在正常离开循环体并且没有 break 语句时才会执行的内容。如果 while 循环中根本没有 break 语句,则 else 子句也会执行。

【例 4-7】 计算 10!的值。

源程序如下:

```
fact=1
i=1
while i<=10:
    fact=fact * i
    i=i+1
print("10!=",fact)
```

本例中的第 1、2 行实现两个变量的初始化,第 3 行进行循环条件判断,第 4、5 行完成阶乘和被乘数的从小到大增量。程序运行后的输出结果为:10!=3628800。

【例 4-8】 有一张厚 1mm 的布,设定面积足够大,能够将它数次对折。问对折多少次,其厚度可以达到或超过珠穆朗玛峰的高度。

求解方法:使用循环结构来表示对折过程,其中 high 和 num 为两个变量,变量 high 表示阶加变量,初始值为 1,每次加上一个“加数”(即 high 本身,不要乘以 2);变量 num 用于统计对折次数,初始值为 1。

源程序如下:

```
num=0
high=1
while high<8848000:                    #珠穆朗玛峰高度(以毫米为单位)
    num=num+1
    high=high +high
print("对折次数: ",num)
print("对应高度: ",high//1000,"米")
```

本例中的第 1、2 行声明变量并进行初始化,第 3～5 行使用阶加法,其中第 4 行统计对折次数,第 5 行计算布的当前厚度,最后输出对折次数和以米为单元的高度。

运行结果如图 4-7 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
对折次数:  24
对应高度:  16777 米
```

图 4-7 例 4-8 的运行结果

【例 4-9】 将任意 9 位的正整数逆序输出。

求解方法:本例中将使用整除和取余运行,以便重复获取个位数,直到 9 个数位均得到时为止。

源程序如下:

```
n=int(input("输入整数: "))
num=0
while n!=0:
    d=n%10                             #获取个位数
```

```
    num=num * 10+d                    #获得部分合成数
    n=n//10                           #缩小 10 倍
print("逆序输出：",num)
```

本例中的第 1 行获得初始数据，第 2 行使 num 的初值为 0。第 3～6 行使用阶加法，其中 d 得到个位数，num 得到部分合成数，n 被 10 整除。循环 9 次后结束且输出结果。

运行结果如图 4-8 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
输入整数：123456789
逆序输出： 987654321
```

图 4-8 例 4-9 的运行结果

【例 4-10】 将 1000 元存入银行，若年利率为 2.75%，计算需多少年后本息共计 1500 元。

源程序如下：

```
rmb=1000
r=0.0275
year=0
while rmb<=1500:
    year=year+1
    rmb=rmb * (1+r)
    print("第",year,"年本息共计",rmb)
```

本例中的第 1～3 行对 3 个变量的初始化。第 4～7 行使用迭代算法，其中 rmb＝rmb *（1＋r）就是迭代公式，第 7 行显示每年的信息。

运行结果如图 4-9 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
第 1 年本息共计 1027.5
第 2 年本息共计 1055.7562500000001
第 3 年本息共计 1084.7895468750003
第 4 年本息共计 1114.621259414063
第 5 年本息共计 1145.2733440479496
第 6 年本息共计 1176.7683610092683
第 7 年本息共计 1209.1294909370233
第 8 年本息共计 1242.3805519377916
第 9 年本息共计 1276.546017116081
第 10 年本息共计 1311.6510325867732
第 11 年本息共计 1347.7214359829095
第 12 年本息共计 1384.7837754724396
第 13 年本息共计 1422.8653292979318
第 14 年本息共计 1461.994125853625
第 15 年本息共计 1502.1989643145998
```

图 4-9 例 4-10 的运行结果

【例 4-11】 分解一个正整数中的全部质因数，例如 90＝2×3×3×5。（提示：质因数是指质数作为因数。）

求解方法：使用穷举算法可以求解问题。使用变量 k 的取值范围为 2～89，逐个由 if 语句判断是否为 n（程序中不是 90）的质因数。若是质因数，则输出变量 k 的值并缩小正整数 n。

源程序如下：

```
n=int(input("输入数据："))
k=2
while k<n:
```

```
    if n%k==0:                          #判断质因数
        print(k,end="×")
        n=int(n/k)                      #修改 n 的值
    else:
        k=k+1                           #修改 k 的值
print(n)                                #输出最后的 n 值
```

本例中的质因数可能重复,所以在找到质因数后不能修改 k 的值,还需要继续循环。

程序运行时键盘输入数据为 281631327,则运行结果如图 4-10 所示。

```
====================== RESTART: D:\Python36\ch4\p1.py ======================
输入数据: 281631327
3×61×61×25229
```

图 4-10　例 4-11 的运行结果

4.3.2　range()函数

要表示等差数列形式的批量数据,可以使用 range()函数,一般调用格式如下:

```
range([<初值>, ] <终值>[, <步长>])
```

range()函数共有 3 个参数,分别表示整数范围的初值(首项)、终值(尾项)和步长(公差)。若省略参数,则默认的＜初值＞为 0,＜步长＞]为 1。当＜终值＞大于＜初值＞时,＜步长＞]应该为正数;而当＜终值＞ 小于＜初值＞时,＜步长＞]应该为负数。

假设用 i 和 j ($i<j$)分别表示范围的＜初值＞和＜终值＞,range(i,j)函数返回一个由整数构成的列表 [i,$i+1$,…,$j-1$]。注意最后一个整数比＜终值＞小 1,也就是说不包含＜终值＞。range()函数和 for 语句经常搭配使用,为 for 语句提供重复操作的次数。

【例 4-12】　显示数列 2,4,6,8。

(1) 由列表直接实现。在 IDLE 交互环境中,输入如下命令:

```
>>>s=[2, 4, 6, 8]
>>>for n in s:print(n, end="  ")
```

运行结果:

```
2  4  6  8。
```

(2) 由 range()函数实现。在 IDLE 交互环境中,输入如下命令:

```
>>>for n in range(2, 10, 2): print(n, end="  ")
```

请读者自行编写由元组直接实现的程序。

4.3.3　for 语句

for 循环的用途很广,常用来构造有限次数的循环结构,例如处理字符串、遍历列表等。for 语句的一般格式如下:

```
for <目标变量>in <对象>:
    <语句块 1>
```

```
    if <条件 1>: break
    if <条件 2>: continue
[else:
    <语句块 2>]
```

for 语句的一般格式中，break、continue、else 等部分都是可选的，这与 while 语句类似。for 语句是针对数据对象中的每个元素(赋值给目标变量)，并执行＜语句块 1＞。

【例 4-13】 计算 5!的值。

求解方法：问题求解具有多样性，计算阶乘也可以由 for 语句实现。

源程序如下：

```
fact=1
for i in range(1,6) : fact=fact*i
print("5!=",fact)
```

本例中的第 2 行自动实现条件判断、阶乘以及从小到大修改循环变量，并执行循环体。

运行结果：

```
5!=120
```

如果修改 for 循环语句中的参数，则可以得到计算阶乘的第二个程序，且实现功能不变。

源程序如下：

```
fact=1
for i in range(10,0,-1) : fact=fact*i
print("10!=",fact)
```

运行结果：

```
10!=3628800
```

【例 4-14】 显示斐波那契数列的前 15 个数。

源程序如下：

```
f1=1
f2=1
print("显示 Fibonacci 数列: ")
print(f1,"\t",f2,end="\t")
for i in range(3,16):
    f=f1+f2
    f1=f2
    f2=f
    print(f,end="\t")
    if i%5==0 : print()
```

本例中的第 1、2 行并对两个变量进行初始化，即 f1＝1 和 f2＝1。第 5～8 行生成斐波那契数列中的后面 13 个数并显示，其中第 6 行生成下一个数，第 7、8 行通过迭代计算出后续循环所需要的数据。注意计数变量 i 的初值为 3，以便保证每行能够输出 5 个数。

运行结果如图 4-11 所示。

```
======================= RESTART: D:\Python36\ch4\p1.py =======================
显示Fibonacci数列:
1         1         2         3         5
8         13        21        34        55
89        144       233       377       610
```

图 4-11　例 4-14 的运行结果

【例 4-15】　使用公式(2/1)×(4/3)×…×(100/99)/4，计算圆周率 π 的近似值。

源程序如下：

```
pi=1/4
for k in range(2,101,2):pi=pi*k/(k-1)
print("pi=",pi)
```

本例中的第 2 行由循环变量 k 对应每个乘积项的分子，并由 50 次乘法操作求得结果。

运行结果：

```
pi=3.1411282254637243
```

【例 4-16】　计算数列 2/1,3/2,5/3,8/5,13/8,21/13,…的前 20 项之和。

求解方法：公式中的分子和分母均是斐波那契数，所以需要使用含迭代运算的 for 循环。

源程序如下：

```
f1=1
f2=2
sum=0
for n in range(1,21,1):
    sum=sum+f2/f1
    f=f1+f2                              #开始 3 步迭代
    f1=f2
    f2=f
print("sum=",sum)
```

本例中的第 1～3 行实现变量初始化，第 6～8 行是迭代运算，以便保证下次循环过程中的数据是正确的。运行程序后的输出结果如下：

```
sum=32.66026079864164
```

【例 4-17】　计算 1＋2!＋3!＋ …＋20!

源程序如下：

```
fact=1
sum=1
for n in range(2,20,1):
    fact=fact*n                          #求和中的通项(指加数)
    sum=sum+fact
print("sum=",sum)
```

运行程序后的输出结果如下：

```
sum=128425485935180313
```

注意：本例可以使用双重循环实现，但效率太低，读者可以自行完成。

【例 4-18】 计算 $s=a+aa+aaa+aaaa+aa\cdots a$(共 n 个 a)，其中 a 是只有个位的正整数。例如 2+22+222+2222+22222。

求解方法：既然循环体是对相同表达式实施重复计算，那么在编程前要找出阶加过程中的“通项”。若设定通项为 term，则通项为 term=10×term+a。初始设置 term 为 0，循环如下：

第 1 次循环：term=10×term+2==10×0+2=2

第 2 次循环：term=10×term+2==10×2+2=22

第 3 次循环：term=10×term+2==10×22+2=222

…

源程序如下：

```
a=int(input("a="))
n=int(input("n="))
term=0                              #表示通项的变量
sum=0
for i in range(1,n+1,1):
    term=10*term+a                  #计算新的通项值
    print("第",i,"项：\t",term)
    sum=sum+term                    #阶加
print("求和结果：\t",sum)
```

本例中的第 1～4 行实现变量初始化，第 6 行计算出每次求和过程中的加数(这是通项公式)，第 8 行进行阶加，结束循环后显示结果。

运行结果如图 4-12 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
a=2
n=5
第 1 项:          2
第 2 项:          22
第 3 项:          222
第 4 项:          2222
第 5 项:          22222
求和结果:          24690
```

图 4-12 例 4-18 的运行结果

【例 4-19】 一个球从 100m 高度自由落下，每次落地后反跳回原高度的一半后再次落下。计算球经过共 100 次落地后，反弹高度合计多少米？第 100 次反弹高度是多少？

源程序如下：

```
h=100
high=0
for n in range(1,101,1):
    h=h/2                           #实除运算
    high=high+h                     #高度的阶加值
print("反弹高度之和：",high)
print("最后反弹高度：",h)
```

运行结果如图 4-13 所示。

```
>>>
======================= RESTART: D:/Python36/ch4/p1.py =======================
反弹高度之和:  100.0
最后反弹高度:  7.888609052210118e-29
```

图 4-13　例 4-19 的运行结果

从运行结果中可以发现,反弹高度之和接近 100(但不是 100),但由于其后的反弹高度值太小,Python 解释器只能默认为 100,这也是一种误差处理。

【例 4-20】 编程找出 100 以内的全部同构数。

同构数是指数本身将出现在对应平方数的右边,例如 5 的平方数为 25,就是同构数。

源程序如下:

```
k=1
for n in range(2,100,1):
    if n<10 and n*n%10==n:              #判断一位数是否为同构数
        print("第",k,"个同构数:",n)
        k=k+1
    if n<100 and n*n%100==n:            #判断两位数是否为同构数
        print("第",k,"个同构数:",n)
        k=k+1
```

本例中的同构数判断是分别以一位数和两位数两种情况进行处理的。

运行结果如图 4-14 所示。

```
======================= RESTART: D:\Python36\ch4\p1.py =======================
第 1 个同构数:  5
第 2 个同构数:  6
第 3 个同构数:  25
第 4 个同构数:  76
```

图 4-14　例 4-20 的运行结果

【例 4-21】 判断正整数是否为质数。

质数是除 1 和自身外没有其他因数的正整数,例如 5 就是一个质数。

求解方法:由键盘输入一个大于 1 的自然数 n,使用变量 k,其取值范围为 $2\sim n-1$,用循环结构来判断所有的 k 值是否是 n 的因数。若是因数,则输出不是质数的信息,并结束程序;若没有任何因数,则输出是质数的信息。

源程序如下:

```
n=int(input("输入数据:"))
k=2
flag=1                                  #设置标志变量
while k<n and flag==1:
    if n%k==0:
        print(n,"不是质数")
        flag==0
        break
    k=k+1
else:
    print(n,"是质数")
```

本例中的循环条件由两项共同控制，其中 flag 是标志变量，初值为 1。若发现不是质数，则将标志变量 flag 的值修改为 0，表示不是质数，从而控制循环结束。若标志变量 flag 的值一致为 1，则表示找到质数。

程序运行两次的结果如图 4-15 所示。

```
====================== RESTART: D:\Python36\ch4\p1.py ======================
输入数据: 113
113 是质数
>>>
====================== RESTART: D:\Python36\ch4\p1.py ======================
输入数据: 115
115 不是质数
```

图 4-15　例 4-21 的运行结果

若将循环条件中的两项分开，同样可以判断质数。

源程序如下：

```
n=int(input("n="))
k=2
flag=0
while k<n:
    if n%k==0:
        flag=1
        break
    k=k+1
if flag==0:
    print(n,"是一个质数")
else:
    print(n,"不是一个质数")
```

第二个程序中的循环结构有两个出口（即退出循环），这是不符合“单入口单出口”的结构化程序原则的，所以建议读者使用第一个程序中的循环结构。

4.3.4　循环嵌套

1. 循环程序组成

通过上述两种循环可以看到，循环程序由 4 个部分组成，如表 4-2 所示。

表 4-2　循环程序组成

组　　成	说　　明
循环初始化	保证循环能够正常执行和重复执行的语句，书写在循环程序的开头
循环控制	保证循环序按规定的循环次数来控制循环条件，以便正常地进行循环
循环工作	作为循环体，完成循环程序中重复的部分
循环修改	保证在整个循环过程中有关的变量能按一定的规律变化

在一个具体循环程序中，上述 4 个部分有时很难明显分开，相互的位置关系可前可后、可相互包含，但是循环的初始化部分一定是书写在循环程序的开始位置。

2. while 循环和 for 循环的比较

(1) 这两种循环都可以处理同一个计算问题，一般情况下它们可以互相代替。

(2) 当使用 while 循环时，循环变量初始化的操作应在 while 语句前完成，而 for 语句可以在 range()函数中完成。

(3) while 循环只能在 while 后面指定循环条件并在循环体中包含使循环趋于结束的语句；for 循环可以在 range()函数中包含使循环趋结束的操作，甚至可以将循环体中的操作全部放到 range()函数中，所以 for 循环的功能更强。

3. 多重循环

多重循环是指在一个循环程序的循环体内又包含着另一个循环，也就是循环嵌套。Python 对循环嵌套的次数并没有任何限制，但是内层循环必须完全嵌套在外层循环中，即不允许交叉和重叠。

for 循环同样支持嵌套，即在一个 for 循环中可以还有其他的任意多个 for 循环，但是不同循环体中的循环变量必须是不同的，这样不但可以提高程序的可读性，也可以避免不必要的错误。

【例 4-22】 求解鸡兔同笼问题。有若干只鸡和兔在同一个笼子里，从笼子上面数，有 35 个头；从下面数，有 94 只脚。求笼中各有多少只鸡和兔？

求解方法：用列方程的方法来求解这个问题。设鸡有 x 只，兔有 y 只，则根据题意列出如下二元一次方程组：

$$\begin{cases} x + y = 35 \\ 2x + 4y = 94 \end{cases}$$

使用双重循环程序。外层循环穷举鸡数，内层循环穷举兔数。在分别计算头数和脚数后，判断是否满足条件，若满足则显示结果。

源程序如下：

```
for x in range(1,36):
    for y in range(1,36):
        head=x+y                    #计算头数
        leg=2 * x+4 * y             #计算脚数
        if head ==35 and leg ==94:  #测试鸡兔同笼条件
            print("鸡数：",x,"\n 兔数：",y)
```

本例中的双重循环分别穷举出鸡、兔的数量，循环次数是 35×35=1225，显然其中含有大量的冗余判断。

运行结果如图 4-16 所示。


```
====================== RESTART: D:/Python36/ch4/p1.py ======================
鸡数： 23
兔数： 12
```

图 4-16　例 4-22 的运行结果

为了减少双重循环导致的过多循环次数，可以修改程序为只有 35 次的单循环，这是由直接计算兔的数量来实现的。

源程序如下：

```
for x in range(1,35):
    y=35-x                         #计算兔的数量
    leg=2*x+4*y
    if leg==94 : print("鸡数: ",x,"\n兔数: ",y)
```

运行该程序后,系统的输出结果与上面程序完全一致。

【例 4-23】 找出 100～200 以内的全部质数。

求解方法:使用双重循环程序。外层循环变量 n 取值范围是 100～200,内层循环变量 y 取值范围为 2～$n-1$,用于对应可能的全部因数。

源程序如下:

```
m=0
print("全部质数: ")
for n in range(100,201,1):
    #判断 n 是否为质数
    k=2
    while k<n:
        if n%k==0 : break          #n 不是质数
        k=k+1
    if k==n:                       #n 是质数
        print(n,end="\t")
        m=m+1
        if m%6==0 : print()
```

运行结果如图 4-17 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
全部质数:
101     103     107     109     113     127
131     137     139     149     151     157
163     167     173     179     181     191
193     197     199
```

图 4-17 例 4-23 的运行结果

由于质数是奇数,所以可以修改程序第 3 行 range()函数的 3 个参数为 101、200 和 2。这样能够减少 50 次循环。另外,变量 m 用于控制每行显示 6 个质数。

【例 4-24】 求解百鸡问题:已知公鸡 5 元 1 只,母鸡 3 元 1 只,小鸡 1 元 3 只,要求用 100 元购得 100 只鸡。

求解方法:这是一个三重循环程序,外层循环变量 x 取值为 1～19,中层循环变量 y 取值为 1～32,内层循环变量 z 取值为 3～99 且增量只能为 3,对其进行穷举并判断。若找出正确结果,则输出其求解。

源程序如下:

```
for x in range(1,20):
    for y in range(1,33):
        for z in range(3,100,3):
            cost=5*x+3*y+z/3       #计算金额
            count=x+y+z            #计算鸡数
```

```
        if cost==100 and count==100:
            print("公鸡数=",x,end="")
            print("\t 母鸡数=",y,end="")
            print("\t 小鸡数=",z)
```

本例是一个三重循环程序,总的循环次数是 19×32×33=20064。

运行结果如图 4-18 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
公鸡数= 4        母鸡数= 18       小鸡数= 78
公鸡数= 8        母鸡数= 11       小鸡数= 81
公鸡数= 12       母鸡数= 4        小鸡数= 84
```

图 4-18　例 4-24 的运行结果

为了减少三重循环导致的过多循环次数,可以修改程序为双重循环,这是通过直接计算小鸡数量来实现的。

源程序如下:

```
for x in range(1,20):
    for y in range(1,33):
        z=100-x-y                          #计算小鸡数量
        cost=5*x+3*y+z/3
        if cost==100:
            print("公鸡数=",x,end="")
            print("\t 母鸡数=",y,end="")
            print("\t 小鸡数=",z)
```

这是一个双重循环程序,外层循环变量 x 取值为 1～19,内层循环变量 y 取值为 1～32,总的循环次数是 19×32=608。运行该程序后,系统的输出结果与上一个程序完全一致。

【例 4-25】 求整数方程 $i^2+j^2=k^2(1<i<j<20)$的全部整数解。

求解方法:使用三重循环对问题的所有可能状态进行测试,直到找到解或将全部可能状态都测试完为止。

源程序如下:

```
for i in range(1,20,1):
    for j in range(1+i,20,1):
        for k in range(1+j,28,1):
            if i*i+j*j==k*k:
                print(i,"*",i,"+",j,"*",j,"=",k,"*",k)
```

本例是一个三重循环,分别穷举出 i,j,k 可能的全部取值,其中 k 的最大值为 28,即 800 的平方根取整。

运行结果如图 4-19 所示。

【例 4-26】 验证角谷猜想。设定正整数 n,如果 n 为偶数,就将它变为 $n/2$,如果为奇数,则将它变为 $3n+1$。不断重复这样的运算,经过有限步后,一定可以得到 1。(详见例 2-21)

求解方法:使用一个双重循环,外层循环分别对应要验证的数据,内层循环验证整数 n

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
3 * 3 + 4 * 4 = 5 * 5
5 * 5 + 12 * 12 = 13 * 13
6 * 6 + 8 * 8 = 10 * 10
8 * 8 + 15 * 15 = 17 * 17
9 * 9 + 12 * 12 = 15 * 15
12 * 12 + 16 * 16 = 20 * 20
```

图 4-19　例 4-25 的运行结果

是否满足角谷猜想。

源程序如下：

```
#用列表表示验证所需的 6 个数据(均是质数)
a=[3,5,7,11,13,17]
for i in range(0,6,1):
    n=a[i]
    print("%2d"%n,"=",end="")
    while n>1:
        if n%2==0:
            n=n//2
        else:
            n=n*3+1
        print("%1d"%n+"→",end="")
    print()
```

运行结果如图 4-20 所示。

```
====================== RESTART: D:\Python36\ch4\p1.py ======================
 3 =10→5→16→8→4→2→1→
 5 =16→8→4→2→1→
 7 =22→11→34→17→52→26→13→40→20→10→5→16→8→4→2→1→
11 =34→17→52→26→13→40→20→10→5→16→8→4→2→1→
13 =40→20→10→5→16→8→4→2→1→
17 =52→26→13→40→20→10→5→16→8→4→2→1→
```

图 4-20　例 4-26 的运行结果

从以上运行结果可以发现，验证角谷猜想的最后 6 个数据分别是 5、16、8、4、2、1。而且若要得到数据 5，则前一个数只能是 10，若要得到 10，则前一个数可以是 3 或 20。

4.3.5　continue、break 和 pass 语句

在 Python 语言中，还可以通过 3 个跳转控制语句来控制程序的执行流程，它们分别是 continue、break 和 pass。

1. continue 语句

continue 语句又称为继续语句，它的功能是结束当前循环，即跳过当前循环中余下的全部语句，接着进行下一次循环。很显然，执行 continue 语句并不会终止循环。continue 语句使程序流程从当前语句跳转至循环的开始处，通常用于跳过循环范围内应该被忽略的语句块。

注意：continue 语句只能用于 while 循环和 for 循环中。

【例 4-27】　输出范围是 100～200 的所有不能被 3 整除的数。

求解方法：对 100～200 的每一个整数进行检查，若不能被 3 整除则输出，否则不输出。

无论是否输出，都要检查下一个数，直到整数 200 被检查到时为止。

源程序如下：

```
count=0
for n in range(100,201,1):
    if n%3==0 : continue
    print(n,end="   ")
    count=count+1
    if count%10==0 : print()
```

本例中的变量 count 用于计数，以便保证输出 10 个数据后进行换行。

运行结果如图 4-21 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
100   101   103   104   106   107   109   110   112   113
115   116   118   119   121   122   124   125   127   128
130   131   133   134   136   137   139   140   142   143
145   146   148   149   151   152   154   155   157   158
160   161   163   164   166   167   169   170   172   173
175   176   178   179   181   182   184   185   187   188
190   191   193   194   196   197   199   200
```

图 4-21　例 4-27 的运行结果

【例 4-28】　用 continue 语句输出乘法九九表。

求解方法：使用双重循环，外循环控制行数，内循环控制列数，但列数不能超过行数。

源程序如下：

```
for i in range(1,10):
    for j in range(1,10):
        if j>i : continue
        k=i*j
        print("%1d"%i+"*%1d"%j+"=%2d"%k,end=" ")
    print()
```

本例中的第 3 行条件表达式 j>i 成立时，将跳过执行内层循环体部分，所以第 4 和 5 行的实际循环次数为 45(即 1+2+3+4+5+6+7+8+9)，而不是 81。

运行结果如图 4-22 所示。

```
======================= RESTART: D:\Python36\ch4\p1.py =======================
1*1= 1
2*1= 2 2*2= 4
3*1= 3 3*2= 6 3*3= 9
4*1= 4 4*2= 8 4*3=12 4*4=16
5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25
6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

图 4-22　例 4-28 的运行结果

2. break 语句

break 语句又称为中断语句，它只能用在循环结构中。使用 break 语句可使流程跳出当前的循环体，从而结束当前正在进行的循环过程。显然，这条语句将直接破坏“单入口单出口”的结构化程序原则，所以这是一条非常不好的语句，虽然如此，但是目前绝大多数语言均没有抛弃它。

注意：break 语句只能用于 while 循环和 for 循环中。

【例 4-29】 break 语句示例。

源程序如下：

```
for r in range(1,11):
    area=3.14 * r * r
    print("半径=",r,"\t 面积=",area)
    if area>100 : break
```

本例中的第 1 行使用一个可能循环 10 次的 for 语句，但在第 6 次循环时，半径 r 为 6，圆面积 area 为 113.0399…，从而导致第 4 行的条件成立且终止循环，所以在实际循环过程只完成 6 次(输出部分也只进行 6 次)，这是循环的非正常退出，而正常退出是 for 循环进行 10 次。

运行结果如图 4-23 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
半径= 1   面积= 3.14
半径= 2   面积= 12.56
半径= 3   面积= 28.259999999999998
半径= 4   面积= 50.24
半径= 5   面积= 78.5
半径= 6   面积= 113.03999999999999
```

图 4-23　例 4-29 的运行结果

【例 4-30】 把整数 316 分成两数之和，其中一个数为 13 的倍数，另一个数为 11 的倍数。

源程序如下：

```
for m in range(13,316,13):
    n=316-m
    if n%11==0:
        print(m,"+",n,"=316")
        break
```

程序中的第 1 行使用"穷举"算法，使循环变量 i 可以遍取所有可能的数据(例如 13，26，39，52，…，312，…)。但在第 4 次循环时，第一个数为 52，另一个数为 264 满足为 11 倍数的条件，所以终止循环过程。运行程序后的输出结果为

```
52 +264 =316
```

使用双重循环也可以求解本题，编写程序如下：

```
for m in range(13,316,13):
    for n in range(11,316,11):
        if n+m==316:
            print(m,"+",n,"=316")
            break
```

本例中的双重循环分别穷举被加数和加数的全部取值，其后判断两者之和是否为 316。这里的 break 语句只是跳出内层循环，所以能够求解出两个答案。

运行结果如图 4-24 所示。

【例 4-31】 有一个阶梯，若每步跨 2 阶，最后余 1 阶；每步跨 3 阶，最后余 2 阶；每步跨 5 阶，最后余 4 阶；每步跨 6 阶，最后余 5 阶；每步跨 7 阶，正好到阶梯顶。问阶梯共有多少阶？

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
52 + 264 =316
195 + 121 =316
```

图 4-24　例 4-30 的运行结果

求解方法：本题又称为爱因斯坦阶梯问题，其现实性表明有解且阶梯数量极为有限。下面使用穷举算法，设定阶梯数量的范围为 2～1000，若能获得解答，则第 1 个解答得到后可使用 break 语句终止程序运行。

源程序如下：

```
for n in range(2,1001,1):
    if n%2==1 and n%3==2 and n%5==4 and n%6==5 and n%7==0 :
        print("爱因斯坦的阶梯数量是",n)
        break
```

本例中，直接写出布尔表达式来表示爱因斯坦阶梯问题，程序运行结果如下：

```
爱因斯坦的阶梯数量是 119
```

3. pass 语句

pass 语句是没有执行操作的语句，通常称为空语句。它的主要用处是，当某个程序段还没有设计好前，可以用 pass 作为占位用的语句，待程序段设计好后再行填入。很明显，下面则是一个含 pass(没有任何操作)的循环程序。

【例 4-32】 pass 语句示例。

源程序如下：

```
for ch in "Hello":
   if ch=="e":
      pass
      print("这是 pass 语句块")
   print("当前字母:",ch)
```

本例中使用 for 循环遍历一个字符串，在发现字符 e 后将执行 pass 语句。这时的 pass 语句只是占位而已，为可能添加的代码提供安放位置。

运行结果如图 4-25 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
当前字母: H
这是pass语句块
当前字母: e
当前字母: l
当前字母: l
当前字母: o
```

图 4-25　例 4-32 的运行结果

4.4　列表处理

分支选择和循环重复是编程中最重要的实现手段，也是体现计算思维概念最多的程序。下面通过一维列表处理、二维列表处理、查找与排序、字符串处理来深入介绍这两类程序的

流程控制。本节介绍列表处理,后面将介绍查找、排序、字符串处理等方面的程序。

在表示大量数据时,数学中可使用向量、行列式、矩阵等,因而许多计算机语言引入一维数组、二维列表和多维数组,但是 Python 没有将数组(array)或称为(matrix)作为内置数据类型,而是提供第三方模块来处理,内含丰富的数学计算。Python 中列表数据的描述能力远比数组强大,下面将分别介绍一维列表和二维列表的处理。

4.4.1 一维列表

访问列表元素是以索引进行的,而索引又是以 1 为步长(增量)有规律变化的,这就为 for 循环实现列表处理提供基础。

【例 4-33】 新建一维列表并逆序显示。

源程序如下:

```
#初始化一维列表并指定长度为 10
lst=[0,0,0,0,0,0,0,0,0,0]
#生成并顺序显示一维列表
print("顺序显示一维列表")
for i in range(0,10,1):
lst[i]=i+1
    print("lst[","%d"%i,"]=%1d"%lst[i],end="  ")
    if i==4 : print()                    #实现每行显示 5 个元素
#逆序显示一维列表
print()
print("逆序显示一维列表")
for i in range(9,-1,-1):
    print("lst[","%d"%i,"]=%1d"%lst[i],end="  ")
    if i==5 : print()
```

运行结果如图 4-26 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
顺序显示一维列表
lst[ 0 ]=1  lst[ 1 ]=2  lst[ 2 ]=3  lst[ 3 ]=4  lst[ 4 ]=5
lst[ 5 ]=6  lst[ 6 ]=7  lst[ 7 ]=8  lst[ 8 ]=9  lst[ 9 ]=10
逆序显示一维列表
lst[ 9 ]= 10, lst[ 8 ]= 9, lst[ 7 ]= 8, lst[ 6 ]= 7, lst[ 5 ]= 6,
lst[ 4 ]= 5, lst[ 3 ]= 4, lst[ 2 ]= 3, lst[ 1 ]= 2, lst[ 0 ]= 1,
```

图 4-26 例 4-33 的运行结果

【例 4-34】 求斐波那契数列 1,1,2,3,5,…的前 20 个数。

求解方法:设定数列的前两个数均为 1,从第 3 个数开始的所有数,其值均为前 2 数之和。另外,程序中还使用变量 i 来控制每行只显示 4 个数。

源程序如下:

```
#初始化一维列表并指定长度为 20,且前 2 个元素均为 1
fib=[1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
#生成斐波那契数列
for i in range(2,20,1):
    fib[i]=fib[i-1]+fib[i-2]
```

```
#显示斐波那契数列
print("显示 Fibonacci 数列",end="")
for i in range(0,20,1):
    if i%4==0 : print()                    #实现每行显示 4 个元素
    print("f[",i,"]=",fib[i],end="\t")
```

运行结果如图 4-27 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示Fibonacci数列
f[ 0 ]= 1       f[ 1 ]= 1       f[ 2 ]= 2       f[ 3 ]= 3
f[ 4 ]= 5       f[ 5 ]= 8       f[ 6 ]= 13      f[ 7 ]= 21
f[ 8 ]= 34      f[ 9 ]= 55      f[ 10 ]= 89     f[ 11 ]= 144
f[ 12 ]= 233    f[ 13 ]= 377    f[ 14 ]= 610    f[ 15 ]= 987
f[ 16 ]= 1597   f[ 17 ]= 2584   f[ 18 ]= 4181   f[ 19 ]= 6765
```

图 4-27　例 4-34 的运行结果

【例 4-35】 求 10 个正整数中的最大值和相应的下标。

求解方法：设定一维列表含有 10 个元素，下标范围为 0～9；变量 most 存放最大值，假设 a[0]为最大值的初值；从 a[1]开始逐个元素与 most 进行比较，若 a[i]>most 则 most 被重新赋值为 a[i]并记录下标 i。

源程序如下：

```
#初始化一维列表并指定长度为 10
a=[1,2,3,4,5,2,4,6,8,0]
most=a[0]
index=0
#求最大值和相应的索引位置
for i in range(1,10,1):
    if a[i]>most :
        most=a[i]
        index=i
#显示 10 个数
print("显示 10 个数")
for i in range(0,10,1) : print(a[i],end=" ")
print()
print("最大值:\t\t",most)
print("索引位置:\t",index)
```

运行结果如图 4-28 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示10个数
1 2 3 4 5 2 4 6 8 0
最大值:         8
索引位置:       8
```

图 4-28　例 4-35 的运行结果

【例 4-36】 使用迭代法计算一个正数的平方根，并将计算数据全部放入一维列表中。

求 $x=\sqrt{a}$ 的迭代公式为 $x_{n+1}=\frac{1}{2}\left(x_n+\frac{a}{x_n}\right)$。

求解方法：可要求相邻两次 x 相减后的绝对值小于 10^{-6} 来控制循环的结束，但本例中

设定循环进行 11 次。

源程序如下：

```
#初始化一维列表并指定长度为 12
a=[0,0,0,0,0,0,0,0,0,0,0,0]
x=int(input("正数:"))
a[0]=x/2                                  #初始化
#使用迭代公式计算生成一维列表并显示
for n in range(0,11,1):
    print("a[",n,"]=\t",a[n])
    a[n+1]=(a[n]+x/a[n])/2                #计算迭代公式
print(x,"的近似平方根：",a[11])
```

运行结果如图 4-29 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
正数:2
a[ 0 ]=  1.0
a[ 1 ]=  1.5
a[ 2 ]=  1.4166666666666665
a[ 3 ]=  1.4142156862745097
a[ 4 ]=  1.4142135623746899
a[ 5 ]=  1.414213562373095
a[ 6 ]=  1.414213562373095
a[ 7 ]=  1.414213562373095
a[ 8 ]=  1.414213562373095
a[ 9 ]=  1.414213562373095
a[ 10 ]=         1.414213562373095
2 的近似平方根：  1.414213562373095
```

图 4-29　例 4-36 的运行结果

从运行结果可以发现，迭代过程的收敛速度非常快，即第 6 次迭代计算就得到结果。

读者可以选择另一个 x 的值查看运行情况，以便了解迭代过程和误差处理情况。

【例 4-37】 已知有整数 2，3，4，5，7，12，14，15，16，19，23，26，编程统计出其中值为 4 的倍数的整数共有多少个。

源程序如下：

```
#初始化一维列表
a=[2,3,4,5,7,12,14,15,16,19,23,26]
m=len(a)                                  #计算一维列表的长度
print("显示一维列表")
for i in range(0,m,1) : print(a[i],end=" ")
#判断并计数
count=0
for i in range(0,m,1) :
    if a[i]%4==0 : count=count+1
print()
print("为 4 倍数的数据个数：",count)
```

运行结果如图 4-30 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
显示一维列表
2 3 4 5 7 12 14 15 16 19 23 26
为4倍数的数据个数：  3
```

图 4-30　例 4-37 的运行结果

【例 4-38】 将含 10 个元素的一维列表按逆序重新存放。

求解方法：从两端分别进行对应元素的交换到正中位置即可，其中左端元素的下标 i 为 0～4，右端元素的下标 $9-i$ 为 9～5，即 a[i]与 a[$9-i$]交换。

源程序如下：

```
#初始化一维列表并指定长度为 10
a=[0,1,2,3,4,5,6,7,8,9]
print("显示一维列表")
for i in range(0,10,1) : print(a[i],end=" ")
#实现逆序存放
for i in range(0,5,1):
    t=a[i]
    a[i]=a[9-i]
    a[9-i]=t;
#显示逆序存放后的一维列表
print("")
print("显示一维列表")
for i in range(0,10,1) : print(a[i],end=" ")
```

运行结果如图 4-31 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示一维列表
0 1 2 3 4 5 6 7 8 9
显示一维列表
9 8 7 6 5 4 3 2 1 0
```

图 4-31　例 4-38 的运行结果

【例 4-39】 用筛法求 100 以内的质数。

筛法思想：给出要筛数据的范围是 2～k，找出 k 以内的质数 $p_1,p_2,p_3,\cdots,p_k$。先用 2 去筛，即把 2 留下，把 2 的倍数全部剔除；再用下一个质数，也就是 3，把 3 留下，把 3 的倍数全部剔除，其中 6，12，…，100 已经在前面被剔除；接下去用下一个质数 5(4 已经剔除)，把 5 留下，把 5 的倍数剔除；不断重复下去，以此类推。实际上，求解 100 以内的质数只需要进行 4 次操作，对应的筛数是 2、3、5 和 7。

求解方法：用一维列表模拟原始筛中的全部数据，即让 a[k]$=k$，所谓剔除合数 k 就是将 a[k]从原值为 k 替换成 0。

源程序如下：

```
import math
#初始化为空的一维列表，以便添加元素
a=[]
x=[0]
#初始化一维列表并指定长度为 101，并且 a[i]=i
for i in range(0,101,1) : a.append(i)
print("显示一维列表")
count=0
for i in range(2,101,1):
```

```
    print(a[i],end="\t")
    count=count+1
    if count%8==0 : print()
#实现筛法
for i in range(2,101,1):
    k=int(math.sqrt(i))
    for i in range(2,k+1,1):
        if a[i]!=0 :                                        #筛选条件
            for j in range(2,100//i+1,1) : a[i*j]=0         #剔除数据
#显示质数
count=0
print("\n 显示全部质数")
for i in range(2,101,1):
    if a[i]!=0:
        count=count+1
        print(a[i],end="\t")
        if count%8==0 : print()
```

运行结果如图 4-32 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示一维列表
2       3       4       5       6       7       8       9
10      11      12      13      14      15      16      17
18      19      20      21      22      23      24      25
26      27      28      29      30      31      32      33
34      35      36      37      38      39      40      41
42      43      44      45      46      47      48      49
50      51      52      53      54      55      56      57
58      59      60      61      62      63      64      65
66      67      68      69      70      71      72      73
74      75      76      77      78      79      80      81
82      83      84      85      86      87      88      89
90      91      92      93      94      95      96      97
98      99      100
显示全部质数
2       3       5       7       11      13      17      19
23      29      31      37      41      43      47      53
59      61      67      71      73      79      83      89
97
```

图 4-32　例 4-39 的运行结果

【例 4-40】　计算身份证中的校验码。

身份证中的 18 位编码分别代表的含义如表 4-3 所示。

表 4-3　身份证编码的含义

位	含　　义	位	含　　义
1～2	省级行政区代码	3～4	地级行政区代码
5～6	县区行政区代码	7～10	出生日期中的年代码
11～12	出生日期中的月代码	13～14	出生日期中的日代码
15～17	顺序码,奇数指男,偶数指女	18	校验码,若是 0～9 则沿用,若是 10 则为 X

17 位数字本体码加权求和公式：

$S = A_i \times W_i, i = 2, \cdots, 18$

i：表示号码字符从右至左包括校验码字符在内的位置序号。

A_i：表示第 i 位置上的身份证号码字符值。

W_i：表示第 i 位置上的加权因子。

i	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
W_i	7	9	10	5	8	4	2	1	6	3	7	9	10	5	8	4	2	1

$Y = S\%11$

计算校验码字符值如下：

Y	0	1	2	3	4	5	6	7	8	9	10
校验码	1	0	X	9	8	7	6	5	4	3	2

源程序如下：

```
ID=510102196109089543                          #设置一个身份证号码
n=ID
A=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]        #初始化列表
#取出身份证号码中的全部数字并存入列表
for i in range(0,18,1):
    A[17-i]=n%10
    n=n//10
print("身份证:\t",A)                            #显示列表中的身份证号码
W=[7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2]        #表示权值的列表
print("权重值:",W)                              #表示权重值
S=0
#计算总和值 S
for i in range(0,17) : S=S+A[i]*W[i]
Y=S%11
print("总和值:\t",S,"\n 校验码:\t",Y)
X=[1,0,10,9,8,7,6,5,4,3,2]                     #表示校验码的列表
if X[Y]==A[17]:
    print("校验位:\t",X[Y])
```

运行结果如图 4-33 所示。

```
======================== RESTART: D:/Python36/ch4/p1.py ========================
身份证:   [5, 1, 0, 1, 0, 2, 1, 9, 6, 1, 0, 9, 0, 8, 9, 5, 4, 3]
权重值: [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
总和值:   328
校验码:   9
校验位:   3
```

图 4-33　例 4-40 的运行结果

注意：由于身份证编码是根据《公民身份号码》(GB 11643—1999)体系定义的，具有法律意义，所以这里的变量名直接沿用，其中含大写字母。

4.4.2　二维列表

二维列表可以由元素是一维列表来表示，其后访问将直接由两个下标确定。通常使用

双重 for 循环，其中外层循环控制行下标 i，内层循环控制列下标 j。

【例 4-41】 将二维列表 a 的全部元素按行、列号互换，并存入二维列表 b。

求解方法：所谓元素按行、列号互换，对应的元素为 a[i][j]与 b[j][i]。

源程序如下：

```
#初始化二维列表并指定长度为 3×3(即 3 行×3 列)
a=[[1,2,3],[4,5,6],[7,8,9]]
b=[[0,0,0],[0,0,0],[0,0,0]]
#显示二维列表 a
print("显示二维列表 a")
for i in range(0,3,1):
    for j in range(0,3,1):
        print(a[i][j],end="\t")
    print()                                     #换行
#生成二维列表 b
for i in range(0,3,1):
    for j in range(0,3,1):
        b[j][i]=a[i][j]
#显示二维列表 b
print("显示二维列表 b")
for i in range(0,3,1):
    for j in range(0,3,1):
        print(b[i][j],end="\t")
    print()                                     #换行
```

运行结果如图 4-34 所示。

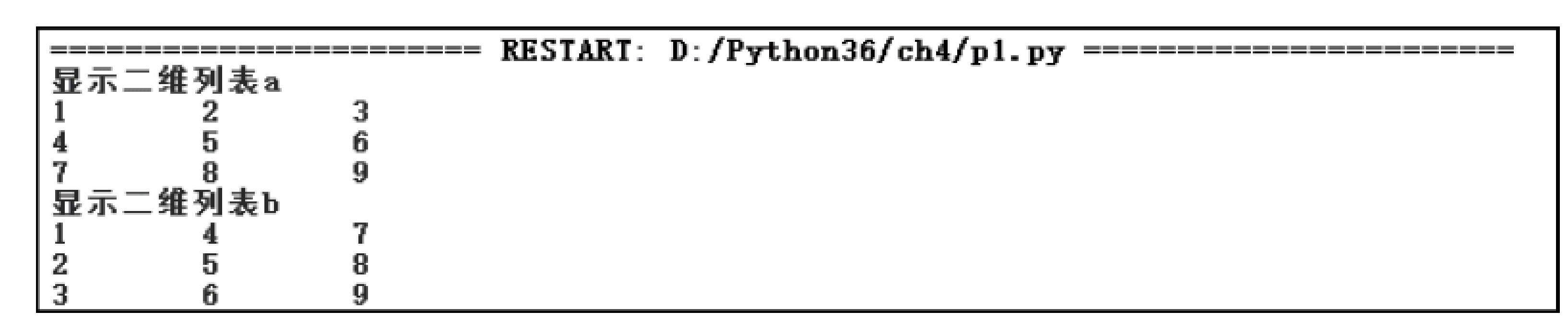

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示二维列表a
1	2	3
4	5	6
7	8	9
显示二维列表b
1	4	7
2	5	8
3	6	9
```

图 4-34　例 4-41 的运行结果

【例 4-42】 有一个 3×4 的二维列表，要求找出最大值以及所在的行列号。

源程序如下：

```
#初始化二维列表并指定长度为 3×4
a=[[2,4,6,8],[9,8,7,6],[1,3,5,7]]
#显示二维列表 a
print("显示二维列表 a")
for i in range(0,3,1):
    for j in range(0,4,1):
        print(a[i][j],end="\t")
    print()                                     #换行
#初始化 3 个变量
row=0
```

```
col=0
most=a[0][0]
#找出最大值以及对应的行、列号
for i in range(0,3,1):
    for j in range(0,4,1):
        if a[i][j]>most :
            most=a[i][j]
            row=i
            col=j
print("最大值=",most)
print("对应行号=",row,"\t 对应列号=",col)
```

运行结果如图 4-35 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示二维列表a
2       4       6       8
9       8       7       6
1       3       5       7
最大值= 9
对应行号= 1     对应列号= 0
```

图 4-35　例 4-42 的运行结果

【例 4-43】 生成一个二维列表,并求主对角线元素之和。

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

求解方法:在沿用 Python 索引体系的前提下,以上的二维列表元素可以直接由如下公式计算而得:$a[i][j]=5i+j+1$。另外,主对角线元素具有行下标与列下标相同的特点。

源程序如下:

```
#初始化二维列表并指定长度为 5×5
a=[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]
#生成二维列表
for i in range(0,5,1):
    for j in range(0,5,1) :
        a[i][j]=5*i+j+1                    #计算元素值
#显示二维列表
print("显示二维列表")
for i in range(0,5,1):
    for j in range(0,5,1):
        print(a[i][j],end="\t")
    print()                                #换行
#求主对角线元素之和
s=0
for i in range(0,5,1) : s=s+a[i][i]
print("主对角线元素之和:",s)
```

运行结果如图 4-36 所示。

```
====================== RESTART: D:\Python36\ch4\p1.py ======================
显示二维列表
1       2       3       4       5
6       7       8       9       10
11      12      13      14      15
16      17      18      19      20
21      22      23      24      25
主对角线元素之和: 65
```

图 4-36　例 4-43 的运行结果

【例 4-44】 显示如下的杨辉三角形(6 行)。

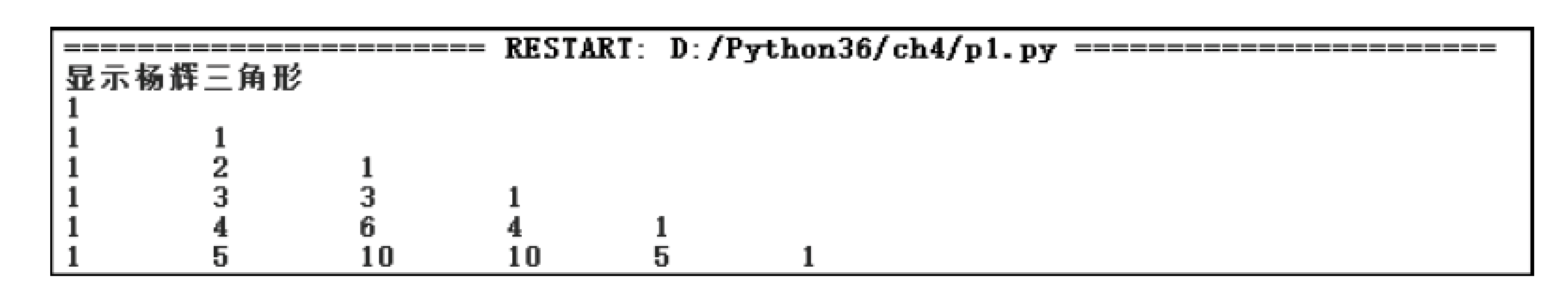
```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
```

求解方法：杨辉三角形具有生成规律，即第 1 列元素和对角线元素均为 1，其余元素可以按公式计算：$b[i][j]=b[i-1][j-1]+b[i-1][j]$。

源程序如下：

```
#初始化二维列表并指定长度为 6×6
a=[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
#初始化首列和对角线元素为全 1
for i in range(0,6,1):
    a[i][0]=1                        #首列元素
    a[i][i]=1                        #对角线元素
#生成杨辉三角形中的其余元素
for i in range(2,6,1):
    for j in range(1,i+1,1):
        a[i][j]=a[i-1][j-1]+a[i-1][j]
#显示杨辉三角形
print("显示杨辉三角形")
for i in range(0,6,1):
    for j in range(0,i+1,1):
        print(a[i][j],end="\t")
    print()                          #换行
```

运行结果如图 4-37 所示。



```
====================== RESTART: D:/Python36/ch4/p1.py ======================
显示杨辉三角形
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
```

图 4-37　例 4-44 的运行结果

【例 4-45】 显示如下的杨辉三角形(5 行)。

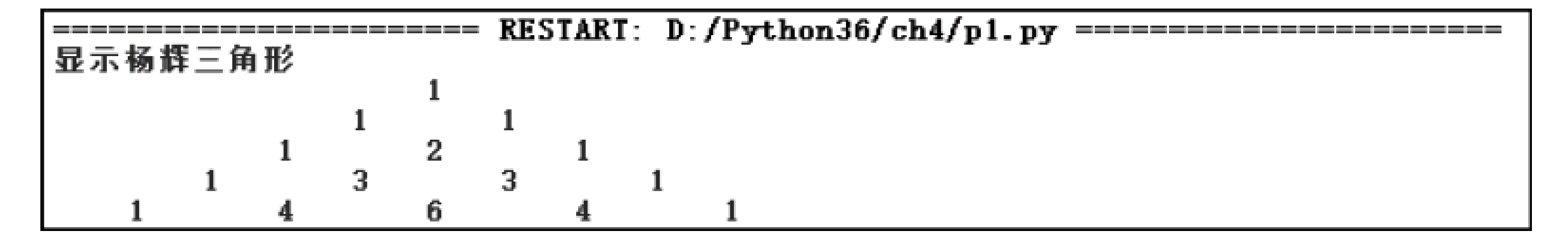
```
                1
            1       1
        1       2       1
    1       3       3       1
1       4       6       4       1
```

求解方法：本例中的二维列表和上例相同，输出效果的不同是如何控制左侧空格的数量。

源程序如下：

```
#初始化二维列表并指定长度为 5×5
a=[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]
#初始化首列和对角线元素为全 1
for i in range(0,5,1):
    a[i][0]=1                              #首列元素
    a[i][i]=1                              #对角线元素
#生成杨辉三角形中的其余元素
for i in range(2,5,1):
    for j in range(1,i+1,1):
        a[i][j]=a[i-1][j-1]+a[i-1][j]
#显示杨辉三角形
print("显示杨辉三角形")
for i in range(0,5,1):
    #显示左边空格串
    for n in range(0,20-4*i,1) : print(end=" ")
    for j in range(0,i+1,1):
        print(a[i][j],end="       ")
    print()                                #换行
```

运行结果如图 4-38 所示。



```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示杨辉三角形
                    1
                1       1
            1       2       1
        1       3       3       1
    1       4       6       4       1
```

图 4-38 例 4-45 的运行结果

【例 4-46】 找出二维列表中每列的最小元素及其所在的行号。

源程序如下：

```
#初始化二维列表并指定长度为 3×3
a=[[8,1,6],[3,5,7],[4,9,2]]
#显示二维列表
print("显示二维列表")
for i in range(0,3,1):
    for j in range(0,3,1):
```

```
        print(a[i][j],end="\t")
    print()                                 #换行
#找出二维列表每列中的最小值及其对应行号
for j in range(0,3):
    #找出第 j 列的最小值及其对应行号
    least=0
    for i in range(1,3):
        if a[i][j]<a[least][j] : least=i
    print("第",j+1,"列最小值",a[least][j],",在第",least+1,"行")
```

运行结果如图 4-39 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示二维列表
8       1       6
3       5       7
4       9       2
第 1 列最小值 3 ,在第 2 行
第 2 列最小值 1 ,在第 1 行
第 3 列最小值 2 ,在第 3 行
```

图 4-39　例 4-46 的运行结果

【例 4-47】 找出一个二维列表中的鞍点,即该元素在该行上最大且列上最小。

求解方法:成为鞍点的要求非常高,一个二维列表可能没有鞍点,也可能有多个鞍点。在初始化二维列表时,可以专门构成有鞍点存在的二维列表以方便程序调试。

源程序如下:

```
#初始化二维列表并指定长度为 5×5,其中含有一个鞍点
a=[[17,35,23,36,12],[11,39,34,36,26],[29,27,1,32,25],[16,22,18,36,28],[14,37,8,
36,19]]
#显示二维列表
print("显示二维列表")
for i in range(0,5,1):
    for j in range(0,5,1):
        print(a[i][j],end="\t")
    print()                                 #换行
#找出二维列表中的鞍点
for i in range(0,5,1):
    #找出第 i 行上的最大元素
    most=a[i][0]
    col=0                                   #记录最大元素所在列号
    for j in range(1,5,1):
        if most<a[i][j]:
            most=a[i][j]
            col=j                           #找出最大元素所在列号
    #判断是否为列上的最小元素
    flag=0                                  #标志变量
    for k in range(0,5,1):
        if most>a[k][col] : flag=1          #不是鞍点
    if flag==0:                             #显示鞍点
```

```
            print("鞍点",a[i][col])              #是鞍点
            print("鞍点在第",i+1,"行第",col+1,"列")
```

运行结果如图 4-40 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
显示二维列表
17      35      23      36      12
11      39      34      36      26
29      27      1       32      25
16      22      18      36      28
14      37      8       36      19
鞍点 32
鞍点在第 3 行第 4 列
```

图 4-40　例 4-47 的运行结果

【例 4-48】　显示幻方。所谓幻方是指行列数相等的一个二维列表，它的每一行、每一列和对角线的元素之和均相等。

例如，3 阶幻方由 1～9 的自然数构成，如图 4-41 所示。

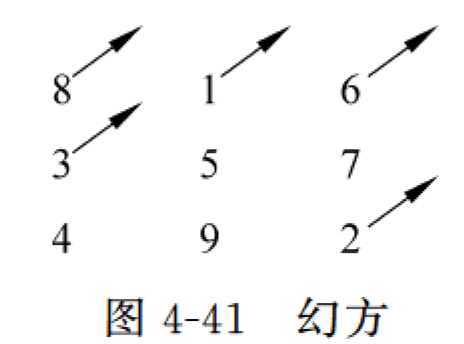


图 4-41　幻方

5 条生成规则如下：

(1) 首行居中位置为 1，其余移动至右上位置，例如数字 1；

(2) 超过首行则移动到最后一行，例如数字 8 和 1；

(3) 超过最后 1 列则移动到首列，例如数字 7 和 2；

(4) 若右上位置的行和列均超界，则移动到下一列，例如数字 6；

(5) 若要填数的位置已经有数则移动到下一列，例如数字 3。

源程序如下：

```
#初始化二维列表并指定长度为 6×6，但第 0 行和第 0 列均不用
a=[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,
0,0,0]]
#设置初始状态，即首行居中位置填 1
i=1
j=5//2+1
a[i][j]=1
#生成 5×5 的幻方
for k in range(2,26):
    i=i-1
    j=j+1
    if i<1 and j>5:                         #表示规则(4)
        i=i+2
        j=j-1
    else:
        if i<1 : i=5                        #表示规则(2)
        if j>5 : j=1                        #表示规则(3)
```

```
        if a[i][j]==0:
            a[i][j]=k                          #表示规则(1)
        else:
            i=i+2
            j=j-1
            a[i][j]=k                          #表示规则(5)
#显示幻方
print("显示 5×5 幻方")
for i in range(1,6,1):
    for j in range(1,6,1):print(a[i][j],end="\t")
    print()
```

运行结果如图 4-42 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示5×5幻方
17      24      1       8       15
23      5       7       14      21
4       6       13      20      0
10      12      19      0       3
11      18      25      2       9
```

图 4-42　例 4-48 的运行结果

请读者自行修改以上程序,分别实现 7 阶和 9 阶幻方。

4.5　查找与排序

查找与排序是计算机中紧密相关的一组操作,快速查找通常要求数据是已经排好序的,而数据排序的主要应用就是查找。

4.5.1　折半查找

查找是在大量数据中找出一个特定的数据。常用的查找是顺序查找和折半查找,显然顺序查找的效率太低,而折半查找要求一维列表有序,是一种效率较高的查找方法。下面只介绍折半查找。

【例 4-49】 有 15 个数按由小到大顺序存放在一维列表中,输入一个数。要求用折半查找法查找该数是列表中的第几个元素。若该数不在列表中,则显示“列表上没有这个数”。

求解方法:首先,将一维列表 a[n]的居中元素与指定值 d 进行比较,若两者相等表示查找成功,则显示其下标位置并退出循环;否则以居中元素为界将一维列表折半分成左、右两部分,如果居中元素大于指定值 d,则进一步查找左半部分,否则进一步查找右半部分。重复以上过程,直到找到满足条件的 d,表示查找成功,或者没有找到表示查找失败。

求解方法:请参考例 2-18。

源程序如下:

```
#初始化一维列表并指定长度为 15
a=[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30]
#显示一维列表
print("显示一维列表",end="")
```

```
for i in range(0,15,1):
    if i%5==0 : print()
    print(a[i],end="\t")
print()
d=int(input("d="))                              #测试程序时输入 5
#设置循环初始状态，即 3 个变量取值
top=0
bot=14
flag=0
while top<bot:
    mid=(top+bot)//2
    if a[mid]==d:                               #表示查找成功
        print("列表上有",d,"这个数")
        flag=1
        break
    elif a[mid]<d:
        top=mid+1                               #表示继续在右半部分查找
    else:
        bot=mid-1                               #表示继续在左半部分查找
if flag==0 : print("列表上没有",d,"这个数")
```

运行结果如图 4-43 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示一维列表
2         4         6         8         10
12        14        16        18        20
22        24        26        28        30
d=5
列表上没有 5 这个数
```

图 4-43　例 4-49 的运行结果

4.5.2　排序

在各种信息检索系统中，支持快速查找算法的基础均是数据有序。目前发展出许多排序算法，例如选择排序、冒泡排序、插入排序、希尔排序、快速排序、归并排序等。下面分别介绍冒泡排序、选择排序和插入排序。

【例 4-50】　使用冒泡法将 10 个数据按照从小到大进行排列。

求解方法：将列表相邻两个数 a[i]和 a[$i+1$]进行比较，当 a[i] $>$ a[$i+1$]时将两个元素互换。第 1 轮比较下来，最大值将放入 a[9]中；接着进行第 2 轮比较，次大数将放入 a[8]中……继续重复上述操作，直到第 9 轮冒泡过程后就将 10 个数按升序存放在列表中。

源程序如下：

```
#初始化一维列表并指定长度为 10
a=[7,8,0,1,4,5,6,2,3,9]
print("显示原始一维列表")
for i in range(0,10,1) : print(a[i],end=" ")
#实现冒泡排序
```

```
for j in range(0,9,1):
    for i in range(0,9-j,1):
        if a[i]>a[i+1]:
            t=a[i]
            a[i]=a[i+1]
            a[i+1]=t
#显示有序一维列表
print()
print("显示有序一维列表")
for i in range(0,10,1) : print(a[i],end=" ")
```

运行结果如图 4-44 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示原始一维列表
7 8 0 1 4 5 6 2 3 9
显示有序一维列表
0 1 2 3 4 5 6 7 8 9
```

图 4-44　例 4-50 的运行结果

【例 4-51】 使用选择法将 10 个数据按照从小到大进行排列。

求解方法：

在[8,6,4,2,0,9,7,5,3,1]中找到最小元素 0,与 a[0]交换；

在[6,4,2,8,9,7,5,3,1]中找到最小元素 1,与 a[1]交换；

在[4,2,8,9,7,5,3,6]中找到最小元素 2,与 a[2]交换；

…

进行 9 次选择过程后,列表就按从小到大排列好了。

源程序如下：

```
#初始化一维列表并指定长度为 10
a=[7,8,0,1,4,5,6,2,3,9]
print("显示原始一维列表")
for i in range(0,10,1) : print(a[i],end=" ")
#实现选择排序
for i in range(0,9,1):
    #找出最小元素下标
    least=i;
    for j in range(i+1,10,1):
        if a[j]<a[least] : least=j
    #交换对应变量的值
    t=a[i]
    a[i]=a[least]
    a[least]=t
#显示有序一维列表
print()
print("显示有序一维列表")
for i in range(0,10,1) : print(a[i],end=" ")
```

运行结果如图 4-45 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示原始一维列表
7 8 0 1 4 5 6 2 3 9
显示有序一维列表
0 1 2 3 4 5 6 7 8 9
```

图 4-45　例 4-51 的运行结果

【例 4-52】 已有一个已排好的一维列表,输入一个数要求按原来排序的规律将它插入一维列表中。

下面介绍两种求解方法。

(1) 第 1 个程序以从右到左进行元素交换来保持有序。

源程序如下:

```
#初始化一维列表并指定长度为 10, 并且 a[9]为插入数据, 最初是-999
a=[1,7,8,17,23,24,59,62,99,-999]
print("显示一维列表")
for i in range(0,9,1):
    print(a[i],end="\t")
    if i==4 : print()
print()
a[9]=int(input("要插入的数据: \t"))              #测试程序时输入 28
#实现插入数据到合理位置
for i in range(9,0,-1):
    #将插入数据交换到前面
    if a[i]<a[i-1]:
        t=a[i-1]
        a[i-1]=a[i]
        a[i]=t;
#显示插入后的有序列表
print("显示插后列表")
for i in range(0,10,1):
    print(a[i],end="\t")
    if i==4 : print()
```

运行结果如图 4-46 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示一维列表
1       7       8       17      23
24      59      62      99
要插入的数据:   28
显示插后列表
1       7       8       17      23
24      28      59      62      99
```

图 4-46　例 4-52 第 1 个程序的运行结果

(2) 第 2 个程序以从左到右进行元素交换来保持有序。

源程序如下:

```
#初始化一维列表并指定长度为 10,并且 a[9]为插入数据
a=[1,7,8,17,23,24,59,62,99,-999]
```

```
print("显示一维列表")
for i in range(0,9,1):
    print(a[i],end="\t")
    if i==4 : print()
print()
a[9]=int(input("要插入的数据: \t"))            #测试程序时输入 48
#实现插入数据到合理位置
for i in range(0,8,1):
    if a[9]<=a[i]:
        t=a[9]
        a[9]=a[i]
        a[i]=t
#显示插入后的有序列表
print("显示插后列表")
for i in range(0,10,1):
    print(a[i],end="\t")
    if i==4 : print()
```

运行结果如图 4-47 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示一维列表
1       7       8       17      23
24      59      62      99
要插入的数据:   48
显示插后列表
1       7       8       17      23
24      48      59      99      62
```

图 4-47　例 4-52 第 2 个程序的运行结果

4.6　字符串处理

字符串作为信息处理的基础，Python 语言支持 string 对象，从而使程序员可以方便地调用 string 对象中的各种函数和常量。不过，这一节是基于计算思维概念构造字符串处理的程序。下面将字符串分为单个字符串和多个字符串分别进行介绍。

4.6.1　单个字符串

【例 4-53】 编程验证一个字符串是否为回文。所谓回文是指一个字符串正读和反读都一样，例如 level、madam 等就是回文。

求解方法：求出输入字符串的长度记为 n，并将其从中央开始分成两部分。其中，左部分下标从小到大为 $0\sim n//2-1$，右部分下标从大到小为了 $n-1\sim n//2$，并对应进行比较。若全部均相同则是回文，否则不是回文。

源程序如下：

```
s=input("输入字符串: ")
n=len(s)
flag=0
```

```
#判断是否回文
for i in range(0,n//2,1):
    if s[i]!=s[n-i-1] :
        flag=1
        break
#选择不同输出
if flag==0:
    print(s,"是回文")
else:
    print(s,"不是回文")
```

程序运行两次的结果如图 4-48 所示。

```
====================== RESTART: D:\Python36\ch4\p1.py ======================
输入字符串: level
level 是回文
>>>
====================== RESTART: D:\Python36\ch4\p1.py ======================
输入字符串: 人人为我, 我为人人
人人为我, 我为人人 是回文
```

图 4-48　例 4-53 的运行结果

从以上运行结果可以发现,Python 字符串处理是包含汉字的。

【例 4-54】 输入一行字符,统计其中的单词数量,单词间用空格分开。

求解方法:从左到右扫描每一个字符,在跑过连续空格后统计单词数量,在跑过连续字符后又处理空格,以此类推。

源程序如下:

```
s=input("输入一行字符:")
num=0
word=0
#统计单词数量
for i in range(0,len(s),1):
    if s[i]==" ":                    #测试是否处理空格(可能连续空格)
        word=0
    else:
        if word==0:                  #测试是否统计单词
            word=1                   #处理字母(通常连续字母)
            num=num+1
print("单词数量:",num)
```

运行结果如图 4-49 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
输入一行字符: I am a student
单词数量:  4
```

图 4-49　例 4-54 的运行结果

【例 4-55】 输入一行字符,分别统计出其中的大写字母、小写字母、阿拉伯数字、空格以及其他字符的个数。

求解方法:从键盘接收一个字符串,通过循环处理字符串中的每一个字符,其中字符串

长度由调用函数得到。

源程序如下：

```
s=input("输入一行字符：")
a=b=c=d=e=0
for i in range(0,len(s),1):
    if s[i]>="A" and s[i]<="Z":            #是大写字母
        a=a+1
    elif s[i]>="a" and s[i]<="z":          #是小写字母
        b=b+1
    elif s[i]>="0" and s[i]<="9":          #是阿拉伯数字
        c=c+1
    elif s[i]==" ":                        #是空格
        d=d+1
    else:
        e=e+1
print(a,b,c,d,e)
```

运行结果如图 4-50 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
输入一行字符：Qwert12345|+-POIUY
6 4 5 0 3
```

图 4-50　例 4-55 的运行结果

【例 4-56】　将一维字符列表 s1 中的全部字符复制到字符一维列表 s2 中。

源程序如下：

```
s1=["This is first string"]
s2=["--------------------"]
for i in range(0,len(s1),1) : s2[i]=s1[i]
print("s1=",s1[0])
print("s2=",s2[0])
```

运行结果如图 4-51 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
s1= This is first string
s2= This is first string
```

图 4-51　例 4-56 的运行结果

注意：只能借助列表而不能直接用字符串，因为字符串是不可变类型。

【例 4-57】　将两个字符串 s1 和 s2 进行比较，若 s1＞s2，输出正数；若 s1＝s2，则输出 0；若 s1＜s2，则输出负数。

源程序如下：

```
s1=input("第 1 个字符串：")
s2=input("第 2 个字符串：")
i=0
#进行比较
```

```
while s1[i]==s2[i] and len(s1)>0:
    i=i+1
#选择输出
if len(s1)==0 and len(s2)==0:
    result=0
else:
    result=ord(s1[i])-ord(s2[i])
print("比较结果: ",result)
```

本例中的第 11 行调用 ord()函数可获得字符所对应的 ASCII 码值,例如字符 A 对应的 ASCII 码值是 65。

运行结果如图 4-52 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
第1个字符串: This
第2个字符串: That
比较结果:  8
```

图 4-52　例 4-57 的运行结果

【例 4-58】 将一个字符串(明文)按规律译成密文,即第一个字母变成第 26 个字母,第 i 个字母变成第 $26-i+1$ 个字母。非字母字符不变,具体转换如下:

```
A→Z         a→z
B→Y         b→y
C→X         c→x
…
Z→A         z→a
```

编程实现将密文转换成明文。

源程序如下:

```
#设置初始密文
s="R drooerhrgXsrmzmvcgdvvp."
print("密文: ",s)
print("明文: ",end="")
#将密文转换成明文
for i in range(0,len(s),1):
    if s[i]>="A" and s[i]<="Z":              #转换大写字母
        print(chr(155-ord(s[i])),end="")
    elif s[i]>="a" and s[i]<="z":            #转换小写字母
        print(chr(219-ord(s[i])),end="")
    else:
        print(s[i],end="")
```

本例中的第 8、10 行调用 chr()函数,与 ord()函数正好相反,可获得 ASCII 码值所对应的字符,例如 ASCII 码的 65 对应的是字符 A。

运行结果如图 4-53 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
密文:  R droo erhrg Xsrmz mvcg dvvp.
明文:  I will visit China next week.
```

图 4-53 例 4-58 的运行结果

4.6.2 多个字符串

为简单起见,在表示多个字符串时可以使用列表数据,其中的列表元素是一个字符串。

【例 4-59】 找出 5 个字符串中的最大者。

求解方法:两个字符串的比较是基于编码,以 ASCII 码为例就是字典序,即排在字典前的字符串小于排在字典后的字符串,例如"This">"That"。

源程序如下:

```
#初始化一维列表
s=["Britain","China","American","Japan","Korea"]
print("显示一维列表:")
for i in range(0,5,1) : print(s[i],end="  ")
most=s[0]
#找出 5 个字符串中的最大者
for i in range(1,5,1):
    if most<s[i] : most=s[i]
print("\n 最大者:",most)
```

运行结果如图 4-54 所示。

```
======================= RESTART: D:/Python36/ch4/p1.py =======================
显示一维列表:
Britain  China  American  Japan  Korea
最大者:  Korea
```

图 4-54 例 4-59 的运行结果

【例 4-60】 统计 20 名成员中姓名以大写字母“M”开头的个数。

求解方法:使用一维列表表示 20 个姓名字符串,而每个姓名字符串中的开头字母由列下标指定,从而使用两个下标,就像二维列表那样。

源程序如下:

```
name=["Peter","Mary","Sharren","Mark","Dicks","Jack","Walt","Disney","Richard",
      "Mara","Charl","Krut","Andi","Jesus","Maya","Maggie","Jim","Edison",
      "Mali","Mex"]
print("全部姓名:")
num=0
for i in range(0,20,1) :
    num=num+1
    print(name[i],end="\t")
    if num%5==0 : print()
count=0
#进行统计
```

```
for i in range(0,20,1):
    if name[i][0]=="M" : count=count+1
print("统计结果:",count)
```

运行结果如图 4-55 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
全部姓名:
Peter   Mary    Sharren Mark    Dicks
Jack    Walt    Disney  Richard Mara
Charl   Krut    Andi    Jesus   Maya
Maggie  Jim     Edison  Mali    Mex
统计结果: 7
```

图 4-55 例 4-60 的运行结果

习 题 4

一、简答题

1. 简述 Python 流程控制语句的分类。
2. 什么是分支选择？简述 Python 分支选择语句的分类。
3. 什么是循环控制？简述 Python 循环控制语句的分类。
4. 简述 range()函数的作用。
5. 什么是循环嵌套？简述 Python 循环程序的组成。
6. 简述 continue 语句、break 语句和 pass 语句的作用和相互间的区别。

二、编程题

1. 将 100 以内的全部质数保存到列表中。
2. 求出 100 以内的全部斐波那契数之和。
3. 求出 10000！中最低位连续多个 0 的数量。
4. 求出一个 7 位正整数的反序数，并显示两者的平方根之和。
5. 已知字符串“ABCD”和“1234”，要求分解成两个列表["A","B","C","D"]和["1","2","3","4"]，并将两个列表合并为一个新列表。
6. 设定列表[123,456,789,135,468]，找出其中的最大反序数。
7. 将一个正整数 n(超过7 位)转换成对应的反序数后，并将全部数位放入列表中。
8. 显示水仙花数。(所谓“水仙花数”是指一个三位数，其各位数字的立方和等于该数本身，例如 $153=1^3+5^3+3^3$。)
9. 老师带 50 位同学去栽树。其中，张老师栽了 5 棵，每位男生栽了 3 棵，每位女生栽了 2 棵，若他们一共栽了 120 棵树，则男生、女生各多少？
10. 约瑟夫问题描述如下：有 n 个人围坐在一个圆桌周围，把这 n 个人依次索引为 1,…,n。从索引是 start 的人开始报数，数到第 num 个人出列，然后从出列的下一个人重新开始报数，数到第 num 个人又出列，…，如此反复直到所有的人全部出列为止。例如当 n=6，start=1，num=5 的时候，出列的顺序依次是 5,4,6,2,3,1。
11. 根据公式 e=1+1+1/2！+1/3!+ …+1/n！+…，计算超越数 e 的值，精确到小

数位的第 6 位。

12. 根据公式 $\sin(x)=x-x^3/3!+x^5/5!-x^7/7!+x^9/9!+\cdots+x^{99}/99!$，计算正弦值。

13. 一个整数加上 100 后是完全平方数，再加上 168 还是完全平方数，找出该数。

14. 统计 1、2、3、4 这 4 个数字能组成几个互不相同且没有重复数字的 3 位数。

第 5 章　函　　数

在程序设计过程中，程序员常会遇到这样的情况，有些操作经常重复出现，有时是在一个程序中多次重复出现，有时是在不同程序中多次重复出现，这些重复运算的程序段是相同的，只不过是每次都以不同的参数进行重复。如果在一个程序中，相同的程序段多次重复编写，势必会使程序行数过多，一方面会占用大量的存储空间，另一方面又浪费宝贵的编程时间。对于许多都要使用的计算函数和求解问题的程序，情况也是如此。如果每个需要这些程序的人都自行单独设计，将浪费大量的时间与金钱，大大地降低程序员的工作效率。解决这一问题的有效办法就是将上述各种情况下共同使用的程序设计成可供其他程序使用的独立程序段。

在 Python 中，这样的程序段被称为函数，其实质就是计算思维中的模块设计概念。对于数值计算、数据库管理、界面布局、多线程、游戏设计等领域的问题，也可以使用函数这一技术。例如，已知正整数 m 和 n，要求计算组合值 $m!/n!/(m-n)!$。在这个应用程序中，要使用计算阶乘的程序 3 次，每次都只对不同的数据进行计算，而程序结构则是一样的。

在 Python 中，函数分为内置函数、标准库函数、第三方库函数和自定义函数四大类。本章主要介绍自定义函数及其调用，内容包括嵌套调用、返回列表、形式参数与实在参数、全局变量与局部变量、lambda 函数、递归函数等。

5.1　函数定义与调用

在 Python 编程规范中，程序书写的顺序通常是，首先进行函数定义，然后才是通过主程序实现函数调用。

5.1.1　函数定义与调用

Python 除了提供大量丰富的内置函数和模块方法，还允许自定义函数，这为编写程序提供了一种有效扩展功能的方法。

1. 函数定义

用户在定义函数时需要使用 def 语句，在 def 语句内部一方面是定义函数的功能，另一方面是将函数值返回给主调函数（即主程序）。

def 语句的一般引用格式如下：

```
def <函数名>(<形参表>):
    <函数体>
    return <返回值>
```

说明：

（1）定义函数由保留字 def 开头，且与其后的<函数名>之间要用空格分隔；

(2) ＜函数名＞必须是合法的 Python 标识符,且其后必须跟圆括号以标识函数;

(3) 圆括号后必须跟冒号,从而让系统自行识别并处理后续的＜函数体＞;

(4) ＜形参表＞位于括号内,表示函数涉及的参数,由逗号分隔不同的形参;

(5) ＜函数体＞用于实现函数的具体执行功能;

(6) ＜函数体＞由语句块组成,必须缩进固定空格数,且最好沿用 IDLE 交互环境中的隐含设置;

(7) 若是有参函数,则要有＜返回值＞选项,可由 return 语句实现。

2. 函数调用及其返回

(1) 函数调用。调用函数的一般格式如下:

```
<函数名>(<实参表>)
```

说明:＜实参表＞表示函数调用时所需的全部参数。由于实参中的值将传递给形参,所以实参应该是常数、有值的变量、表达式、另一个函数调用等。

(2) 返回语句。return 语句的功能是从被调函数返回到主调函数继续执行,返回时可以附带一个返回值,由 return 语句后面的参数指定。其一般引用格式如下:

```
return  <表达式>
```

说明:

① ＜表达式＞用于表示函数的返回值。

② 定义函数必须以 return(＜表达式＞)语句结尾,除非没有返回值。

③ 若有可选项＜表达式＞,则函数将返回该表达式的值;若无可选项＜表达式＞),则系统自动返回常量 None。

一个函数可以通过 return 语句返回一个特定类型的值,也可以不使用 return 语句,而是只执行函数主体中的代码,这种情况下,函数将向调用者返回一个未定义的值。实际上,创建函数的方法与编写其他应用程序是完全相同的,其差别仅仅是在一个函数中至少要有一条 return 语句,除非没有返回值。

(3) 函数调用过程。当函数调用出现在主程序中时,将执行所指定的 a()函数。在 a()函数执行到 return 语句时,将流程转移主程序,同时将返回值传递给主程序,然后继续执行函数调用后面的语句。函数调用的过程如图 5-1 所示。

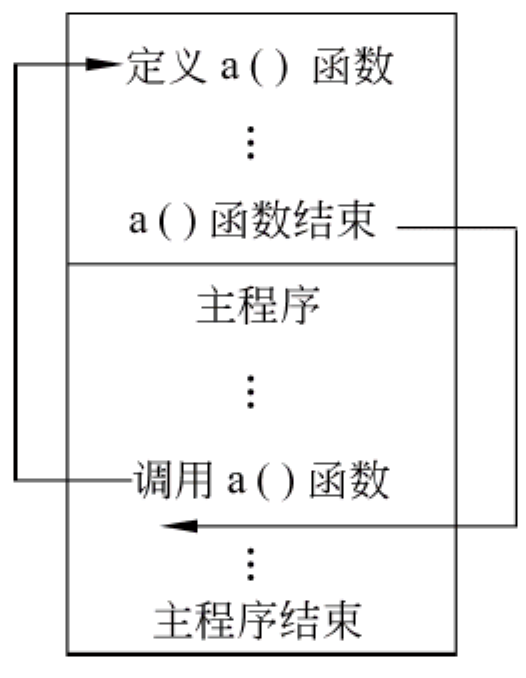


图 5-1　**函数调用过程**

3. 无参函数及其调用

无参函数在没有参数的情况下,使函数没有通用性,即运行结果是固定的。一般用于显示指定信息,若涉及数据也只是转换,而不是计算,所以没有返回值。

【例 5-1】 显示 3 行提示信息。

源程序如下:

```
#定义两个无参函数
def printstar():
    print("******************************")
def print_message():
```

```
    print("*    Computation Thinking!    *")
#主程序并 3 次调用无参函数
printstar()
print_message()
printstar()
```

本例中定义两个无参函数，主程序将三次调用无参函数。

运行结果如图 5-2 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
*****************************
*   Computation Thinking!   *
*****************************
```

图 5-2　例 5-1 的运行结果

【例 5-2】　显示一个平行四边形图案。

源程序如下：

```
#定义无参函数
def printstar():
    s="*****************"
    for i in range(0,7,1):
        #显示左侧空格
        for k in range(0,i+3,1):
            print(" ",end="")
        print(s)                                    #显示星号串
#主程序并调用无参函数
printstar()
```

本例中定义无参函数只是显示图案，所以主程序直接调用，并没有参数传递和变量引用。

运行结果如图 5-3 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
   *****************
    *****************
     *****************
      *****************
       *****************
        *****************
         *****************
```

图 5-3　例 5-2 的运行结果

4. 有参函数及其调用

【例 5-3】　找出两个数中的较大数。

源程序如下：

```
#定义有参函数
def larger(x,y):
    if x>y:
        return x
    else:
        return y
```

```
#主程序及函数调用
a=int(input("输入第 1 个数: "))
b=int(input("输入第 2 个数: "))
c=larger(a,b)
print("较大数: \t",c)
```

本例中的有参函数 larger()在获得由实参 a 和 b 传递的数据后,使函数能够找出指定两数中的较大数,从而使函数具有通用性,即运行结果不是固定的。

运行结果如图 5-4 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入第1个数: 8
输入第2个数: 5
较大数:	8
```

图 5-4　例 5-3 的运行结果

5.1.2　嵌套调用

在执行主程序时可以调用第一个函数,在执行第一个函数时还可以调用第二个函数,这样可以一个又一个地调用下去,这种调用称之为嵌套调用。下面用一个例子加以说明。

【例 5-4】　计算 1+(1+2)+…+(1+2+3+4+5+6+7+8+9+10)。

求解方法:由于这里的和式由 20 项阶加组成,但每一项又是一个需若干数据阶加的式子。所以要定义两个函数,并进行嵌套调用。

源程序如下:

```
#定义函数 sum1()对一项阶加
def sum1(n):
    sum=0
    for i in range(1,n+1,1) : sum=sum+i
    return sum
#定义函数 sum2()对若干数据累加
def sum2(n):
    sum=0
    for i in range(1,n+1,1):
        #第二次调用函数
        sum=sum+sum1(i)
    return sum
#主程序并嵌套调用函数
n=int(input("输入数据: "))
#第一次调用函数
s=sum2(n)
print("累加和值: ",s)
```

本例中的嵌套调用过程是,主程序调用 sum2()函数,sum2()函数再调用 sum1()函数,从而构成两重嵌套的调用函数过程。

运行结果如图 5-5 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
输入数据: 10
累加和值:  220
```

图 5-5　例 5-4 的运行结果

5.1.3　返回值类型与函数类型

由于 Python 使用解释方式实现语义分析，进而导致变量的数据类型可以动态变化。所以，实参类型、返回值类型和函数类型均可以动态变化。

【例 5-5】　实参类型动态变化示例。

源程序如下：

```
#定义函数
def larger(x,y):
    if x>y:
        return x
    else:
        return y
#主程序及其 4 次调用函数
a=3.5
b=4.5
c=larger(a,b)
print(a,b,"中的较大数：",c)
m=6
n=2
d=larger(m,n)
print(m,n,"中的较大数：",d)
s1="This"
s2="That"
e=larger(s1,s2)
print(s1,s2,"中的较大数：",e)
c1=3+4J
c2=4-5J
f=larger(c1,c2)
print(c1,c2,"中的较大数：",f)
```

本例中进行 4 次函数调用，第 1 次是 2 个实数作为实参，第 2 次是 2 个整数作为实参，第 3 次是 2 个字符串作为实参，第 4 次是 2 个复数作为实参。运行结果表明，前 3 次调用都得到正确结果，但是复数不能进行大于比较，因此系统将显示错误代码。

运行结果如图 5-6 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
3.5 4.5 中的较大数:  4.5
6 2 中的较大数:  6
This That 中的较大数:  This
Traceback (most recent call last):
  File "D:/Python36/ch5/p1.py", line 22, in <module>
    f=most(c1,c2)
  File "D:/Python36/ch5/p1.py", line 3, in most
    if x>y:
TypeError: '>' not supported between instances of 'complex' and 'complex'
```

图 5-6　例 5-5 的运行结果

5.1.4 返回列表

让函数返回一个列表，并由列表中的多个元素来描述函数的多值计算，能够解决函数式程序设计中只能实现单值返回的问题。

【例 5-6】 编程生成由 6 个随机数构成的列表。

源程序如下：

```
#导入 random 模块
import random
#定义函数生成由 n 个随机数构成的列表
def my_random(n):
    a=[]
    for k in range(n):
        #将一个随机数附加到列表中
        a.append(random.random())
    return a
#主程序及其调用函数
n=int(input("输入数据："))
m=my_random(n)
print("列表元素：")
#输出列表中的所有元素
for k in m: print(k)
```

运行结果如图 5-7 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入数据：6
列表元素：
0.9641217898452976
0.009672201050304974
0.2481433152076259
0.026247813799271658
0.7906652501022456
0.4408959551431382
```

图 5-7 例 5-6 的运行结果

5.2 形式参数与实在参数

Python 的函数与其他高级语言一样，主程序把一些参数传递给函数，在函数对这些参数进行运算和处理后，再把结果传递给主程序。参数传递是通过主程序中的实在参数和被调用函数中的形式参数相结合来完成的，换句话说，在主程序中要列出实在参数，而在函数中要设置一些形式参数。当进行函数调用时，使形式参数和实在参数相结合，从而实现参数的传递。形式参数(简称形参)和实在参数(简称实参)相结合的具体情况如下：

(1) 实参和形参在数量上最好相等；

(2) 对应位置上的实参和形参最好数据类型一致或兼容；

(3) 实参可以是常量、变量、表达式或另一个函数，而形参只能是变量；

(4) 如果实参为常量，则将常量赋值给形参；

(5) 如果实参为表达式,则将表达式的值赋值给形参;
(6) 如果实参为另一个函数,则一定要能够获得返回值;
(7) 如果实参为已赋值变量,则将其值赋给形参。

5.2.1 简单变量作为实参

【例 5-7】 设定两个正整数 m 和 n,计算组合值 $m!/n!/(m-n)!$。

求解方法:本例将定义计算阶乘的函数,其中实参是有值的简单变量,其值由系统自行赋给形参,主程序通过 3 次调用计算阶乘的函数来获得组合值。

源程序如下:

```
#定义有参函数 comp()
def comp(k):
    fact=1
    for i in range(1,k+1,1) : fact=fact*i
    return fact
#主程序
m=int(input("m="))
n=int(input("n="))
#三次调用函数 comp()
c=comp(m)/comp(n)/comp(m-n)
print("6!/2!/4!=",c)
```

运行结果如图 5-8 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
m=6
n=2
6!/2!/4!= 15.0
```

图 5-8 例 5-7 的运行结果

【例 5-8】 利用穷举算法找出两个正整数中的最大公约数。

求解方法:获取两数中的较小数并由变量 least 表示,使用 for 循环穷举可能的因数,即变量 k 的取值范围(必须降序)为 least~1。若 x、y 同时能够整除 k,则表示 k 是最大公约数,将其值返回给主程序。由于 k 的取值过程是从大到小的,所以首次条件成立时就是最大公约数。

源程序如下:

```
#定义函数 gcd(x, y)找出两数中的最大公约数
def gcd(x,y):
   #获取两数中的较小数
   if x>y:
       least=y
   else:
       least=x
   for k in range(least,0,-1):
       #若 x, y 同时整除 k, 则 k 是最大公约数
       if(x%k==0 and y%k==0):
```

```
            gcd=k
            return gcd
#主程序
a=int(input("a="))
b=int(input("b="))
#调用函数并显示结果
print(a,"和",b,"的最大公约数为",gcd(a,b))
```

运行结果如图 5-9 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
a=28
b=16
28 和 16 的最大公约数为 4
```

图 5-9　例 5-8 的运行结果

【例 5-9】 利用欧几里得算法找出两个正整数中的最大公约数。

求解方法：欧几里得算法又称辗转相除法。设定两个正整数为 x 和 y，若 $x\%y$ 的值大于 0，则重复进行如下 3 个操作：$r=x\%y$，$x=y$，$y=r$，直到 $x\%y$ 为 0 时，这时的 y 就是最大公约数。

源程序如下：

```
#定义函数 gcd(x, y)找出两数中的最大公约数
def gcd(x,y):
    r=x%y
    while r>0:
        x=y
        y=r
        r=x%y
    return y
#主程序及其调用函数
a=int(input("a="))
b=int(input("b="))
print(a,"和",b,"的最大公约数为",gcd(a,b))
```

【例 5-10】 验证哥德巴赫猜想。猜想的大意是，任意大于等于 6 的偶数都可以写成两个质数之和，例如：6＝3＋3，8＝3＋5，10＝3＋7。编程验证 100 以内的所有偶数。

求解方法：定义 $p(m)$ 函数，功能是若 m 为质数，则函数返回值为 1，否则返回值为 0。主程序使用 for 循环穷举所有偶数，并两次调用 $p(m)$ 函数进行是否质数的判断。另外，使用变量 count 控制每行显示 5 个偶数。由于至今人类并没有发现反例，所以验证过程能够正常进行。

源程序如下：

```
#定义函数 p(m), 若取值为 1 则是质数, 否则取值为 0 表示不是质数
def p(m):
    k=2
    flag=0
    while k<m:
```

```
        if m%k==0:                          #不是质数
            flag=1
            break
        k=k+1
    if flag==0:
        return 1
    else:
        return 0
#主程序及其两次调用函数
count=0                                     #控制换行
for n in range(6,100,2):                    #穷举全部偶数
    for k in range(3,n,1):                  #穷举可能的质数
        if p(k)==1 and p(n-k)==1:
            count=count+1
            print(n,"=",k,"+",n-k,end="\t")
            if count%4==0 : print()
            break
```

运行结果如图 5-10 所示。

```
======================== RESTART: D:/Python36/ch5/p1.py ========================
6 = 3 + 3       8 = 3 + 5       10 = 3 + 7      12 = 5 + 7
14 = 3 + 11     16 = 3 + 13     18 = 5 + 13     20 = 3 + 17
22 = 3 + 19     24 = 5 + 19     26 = 3 + 23     28 = 5 + 23
30 = 7 + 23     32 = 3 + 29     34 = 3 + 31     36 = 5 + 31
38 = 7 + 31     40 = 3 + 37     42 = 5 + 37     44 = 3 + 41
46 = 3 + 43     48 = 5 + 43     50 = 3 + 47     52 = 5 + 47
54 = 7 + 47     56 = 3 + 53     58 = 5 + 53     60 = 7 + 53
62 = 3 + 59     64 = 3 + 61     66 = 5 + 61     68 = 7 + 61
70 = 3 + 67     72 = 5 + 67     74 = 3 + 71     76 = 3 + 73
78 = 5 + 73     80 = 7 + 73     82 = 3 + 79     84 = 5 + 79
86 = 3 + 83     88 = 5 + 83     90 = 7 + 83     92 = 3 + 89
94 = 5 + 89     96 = 7 + 89     98 = 19 + 79
```

图 5-10　例 5-10 的运行结果

【例 5-11】 找出 100 以内两两相邻的全部质数，例如 3 和 5，5 和 7，11 和 13，等等。

源程序如下：

```
#定义函数
def p(m):
    k=2
    flag=0
    while k<m:
        if m%k==0:
            flag=1
            break
        k=k+1
    if flag==0:
        return 1
    else:
        return 0
#主程序及其两次调用函数
count=0
```

```
for n in range(3,100,2):                    #穷举全部奇数
    if p(n)==1 and p(n+2)==1:              #是否是相邻质数
        count=count+1
        print("第",count,"对相邻质数：",n,":",n+2)
```

主程序使用 for 循环穷举所有奇数，并两次调用 p(n)和 p(n+2)函数进行是否质数的判断。

运行结果如图 5-11 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
第 1 对相邻质数： 3 : 5
第 2 对相邻质数： 5 : 7
第 3 对相邻质数： 11 : 13
第 4 对相邻质数： 17 : 19
第 5 对相邻质数： 29 : 31
第 6 对相邻质数： 41 : 43
第 7 对相邻质数： 59 : 61
第 8 对相邻质数： 71 : 73
```

图 5-11　例 5-11 的运行结果

【例 5-12】 找出 500 以内的全部亲密数对。

如果整数 A 的全部因数(包括 1，不包括 A 本身)之和等于 B，且整数 B 的全部因数(包括 1，不包括 B 本身)之和等于 A，则将整数 A 和 B 称为亲密数对。例如 220 和 284 就是一对亲密数，计算过程如下：

(1) 220 ＝1＋ 2＋ 4＋ 5＋ 10＋ 11＋ 20＋ 22＋ 44＋ 55＋ 110＝ 284

(2) 284 ＝1＋ 2＋ 4＋ 71＋ 142＝ 220

源程序如下：

```
#定义函数 p()求 n 的因数和
def p(n):
    s=0
    for k in range(1,n,1):
        if n%k==0 : s=s+k
    return s
#定义函数 display()显示 n 的因数阶加式
def display(n):
    print(n,"=1",end="")
    for k in range(2,n,1):
        if n%k==0 : print("+",k,end="")
#主程序及其两次调用函数
for a in range(1,500,1):
    for b in range(a+1,500,1):
        if p(a)==b and p(b)==a:             #测试是否亲密数对
            display(a)
            print("=",b)
            display(b)
            print("=",a)
            print("亲密数：",a,b)
            break
```

主程序使用双重循环以穷举可能的 A 和 B，然后构成判断亲密数对的条件，若成立则显示亲密数对。由于亲密数对的构成条件太高，本例程序只能得到一组答案。另外，display()函数具有参数，但没有返回值，只是内部进行显示数据操作。

运行结果如图 5-12 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
220 =1+ 2+ 4+ 5+ 10+ 11+ 20+ 22+ 44+ 55+ 110= 284
284 =1+ 2+ 4+ 71+ 142= 220
亲密数:  220 284
```

图 5-12　例 5-12 的运行结果

【例 5-13】 找出满足如下条件的整数 n：设定 n 是一个 4 位数，它的 9 倍恰好是从右到左读的反序数。

求解方法：由于一个 4 位数 9 倍后的最大数是 9999，所以 n 的取值范围是 1000～1111。问题求解时可以利用 for 循环进行穷举，并逐个进行处理。

源程序如下：

```
#定义函数
def rev(n):
    m=n
    d=0                                          #表示反序数
    for i in range(1,5,1):
        d=d*10+m%10
        m=m//10
    return d
#主程序并调用函数
for i in range(1000,1111,1):
    if 9*i==rev(i):
        print(i,"是一个反序数")
```

本例中的 rev()函数用于定义如何获得反序数，循环 4 次是将 m 的个位数合成反序数，m 被整除 10 直到为 0 时结束循环。

运行结果：1089 是一个反序数。

【例 5-14】 验证 6174 猜想。它是由印度数学家设计卡普耶卡在 1955 年发现的。猜想的大意是，对 4 位数的进行变换，规则如下：任意给出一个 4 位数 k_0，用它的 4 个数位(个位、十位、百位、千位)由大到小重新排列成一个 4 位数 m(即最大数)，再减去它的反序数 rev(m)(即最小数)，得出数 k_1；然后，继续对 k_1 重复上述变换，得出数 k_2……如此进行下去，无论 k_0 是多大的 4 位数，只要 4 个数字不全相同，最多进行 7 次上述变换，就会出现 4 位数 6174。例如 k_0 是 5298，则可以通过 6 次变换得到 6174。

(1) k_1＝9852-2589＝7263

(2) k_2＝7632-2367＝5265

(3) k_3＝6552-2556＝3996

(4) k_4＝9963-3699＝6264

(5) k_5＝6642-2466＝4176

(6) k_6＝7641-1467＝6174

注意：由于 6174＝7641－1467，所以变换过程结束。

求解方法：首先将一个 4 位数中的 4 个数位分别取出并存放至列表变量中，然后将列表排序后合成最大数和最小数，最后计算出新的 4 位数。若这个 4 位数不是 6174，则继续循环，否则退出循环。

源程序如下：

```
#定义函数
def magic_number(k):
    d=[0,0,0,0]
    i=1
    while k!=6174:
        d[0]=k//1000                                    #计算千位
        d[1]=k//100%10                                  #计算百位
        d[2]=k//10%10                                   #计算十位
        d[3]=k%10                                       #计算个位
        d.sort()                                        #将列表从小到大排序
        m_pos=d[3]*1000+d[2]*100+d[1]*10+d[0]           #合成最大数
        m_rev=d[0]*1000+d[1]*100+d[2]*10+d[3]           #合成最小数
        k=m_pos-m_rev                                   #计算下一个数
        print("第",i,"次计算过程:\t",k,"=",m_pos,"-",m_rev)
        i=i+1
#主程序及其调用
d=int(input("d ="))
magic_number(d)
```

本例中的 magic_number(k)函数用于验证 6174 猜想，其中列表 d 用于表示 4 位数中的 4 个数位，并将其升序排列后去合成最大数，反序排列后去合成最小数，进而计算下一个 4 位数。

设定输入 5298，运行结果如图 5-13 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
d=5298
第 1 次计算过程:   7263 = 9852 - 2589
第 2 次计算过程:   5265 = 7632 - 2367
第 3 次计算过程:   3996 = 6552 - 2556
第 4 次计算过程:   6264 = 9963 - 3699
第 5 次计算过程:   4176 = 6642 - 2466
第 6 次计算过程:   6174 = 7641 - 1467
```

图 5-13　例 5-14 的运行结果

【例 5-15】 验证 495 猜想。与 6174 猜想对应的是 495 猜想，它是对任意 3 位数的一种变换，所以可改写程序如下。

源程序如下：

```
#定义函数
def magic_number(k):
    d=[0,0,0]
    i=1
    while k!=495:
```

```
        d[0]=k//100
        d[1]=k//10%10
        d[2]=k%10
        d.sort()
        m_pos=d[2] * 100+d[1] * 10+d[0]
        m_rev=d[0] * 100+d[1] * 10+d[2]
        k=m_pos-m_rev
        print("第",i,"次计算过程:\t",k,"=",m_pos,"-",m_rev)
        i=i+1
#主程序及其调用
d=int(input("d="))
magic_number(d)
```

设定输入 529,运行结果如图 5-14 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
d=529
第 1 次计算过程:  693 = 952 - 259
第 2 次计算过程:  594 = 963 - 369
第 3 次计算过程:  495 = 954 - 459
```

图 5-14　例 5-16 的运行结果

注意:由于 495＝954－459,所以变换过程结束。

5.2.2　一维列表作为实参

虽然一维列表含有许多元素,但 Python 是将列表作为一个对象来处理的,即只是一个参数。下面将一维列表分为两种情况进行介绍,一是一维列表整体作为实参,二是一维列表中的元素作为实参。

1. 一维列表整体作为实参

【例 5-16】 找出 10 个数中的最大数。

源程序如下:

```
#定义函数
def most(b):
    m=b[0]
    for i in range(1,9):
        if m<b[i] : m=b[i]
    return m
#主程序及其调用函数
#创建一维列表并初始化
a=[1,2,3,4,5,0,9,8,7,6]
print("全部数: ",end="")
for i in a : print(i,end="  ")
print()
print("最大数: ",most(a))
```

本例中的 10 个数是利用一维列表来模拟的,这样能够减轻数据输入的问题。

运行结果如图 5-15 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
全部数: 1  2  3  4  5  0  9  8  7  6
最大数:  9
```

图 5-15　例 5-16 的运行结果

2. 一维列表中的元素作为实参

【例 5-17】 有两个一维列表 a 和 b,各有 10 个元素,将它们对应地逐个进行比较(即 a[0] 与 b[0] 比较,a[1]与 b[1] 比较,以此类推)。若 a 中的元素大于 b 中的相应元素的数目多于 b 中的元素大于 a 中相应元素的数目(例如,a[i]>b[i]是 6 次,b[i]>a[i]是 3 次,其中 i 每次为不同的值,则认为 a 大于 b),并分别统计出两个一维列表相应元素大于、等于、小于的次数。

源程序如下:

```
#定义函数
def large(x,y):
    if x>y:
        return 1
    elif x<y:
        return -1
    else:
        return 0
#主程序及其调用函数
#创建列表并初始化
a=[3,5,7,9,8,6,4,2,0,0]
b=[3,8,9,-1,-3,5,6,0,4,0]
print("列表 a: ",end="")
for i in a : print(i,end="  ")
print()
print("列表 b: ",end="")
for i in b : print(i,end="  ")
print()
n=m=k=0
for i in range (0,len(a),1):
    if large(a[i],b[i])==1:
        n=n+1
    elif large(a[i],b[i])==0:
        m=m+1
    else:
        k=k+1
print("a[i]>b[i]:",n)
print("a[i]=b[i]:",m)
print("a[i]<b[i]:",k)
if n>k:
    print("列表 a 大于列表 b")
elif n<k:
```

```
    print("列表 a 小于列表 b")
else:
    print("列表 a 等于列表 b")
```

本例中定义 large(x,y)函数判断两数的大于、等于和小于关系。主函数利用两个列表来描述数据,其后调用 large(x,y)函数并进行统计。

运行结果如图 5-16 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
列表a: 3  5  7  9  8  6  4  2  0  0
列表b: 3  8  9  -1  -3  5  6  0  4  0
a[i]>b[i]: 4
a[i]=b[i]: 2
a[i]<b[i]: 4
列表a等于列表b
```

图 5-16　例 5-17 的运行结果

5.2.3　二维列表作为实参

为扩大程序的描述能力和应用范围,在 Python 中还可以引入二维列表作为实参。

【例 5-18】　有一个 3×4 的二维列表,找出其中的最大值。

源程序如下:

```
#定义函数
def mst(a):
    most=a[0][0];
    for i in range(0,3,1):
        for j in range(0,4,1):
            if a[i][j]>most : most=a[i][j]          #求出最大值
    return most
#主程序及其调用函数
a=[[1,3,5,7],[2,4,6,8],[15,17,34,12]]
print("列表: ")
for i in range(0,3,1):
    for j in range(0,4,1):
        print(a[i][j],end="\t")
    print()
print("最大值: ",mst(a))
```

运行结果如图 5-17 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
列表:
1       3       5       7
2       4       6       8
15      17      34      12
最大值: 34
```

图 5-17　例 5-18 的运行结果

5.2.4　可变参数

可变参数是指形参前面有星号,从而使形参成为可变参数,这样就可以使主程序在调用

函数时可以书写更多的实参，这样能够扩展程序的功能。

【例 5-19】 利用可变参数计算 $1+2+3+\cdots+n$，其中 n 是任意正整数。

源程序如下：

```
#定义函数计算若干连续正整数的阶加
def sum(a,b, * c):
    s=a+b
    for n in c : s=s+n
    return s
#主程序及其 3 次调用函数
print("1+2+3=",sum(1, 2,3))                    #计算 1+2+3
print("1+2+3+4=",sum(1,2,3,4))                 #计算 1+2+3+4
print("1+2+3+4+5=",sum(1,2,3,4,5))             #计算 1+2+3+4+5
```

本例中的 sum()函数引用可变参数 c，其标识就是符号“ * ”，从而使函数调用中的实参书写更灵活更全面。

运行结果如图 5-18 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
1+2+3= 6
1+2+3+4= 10
1+2+3+4+5= 15
```

图 5-18　例 5-19 的运行结果

5.3　变量的作用域

变量的作用域是指变量从定义、使用到最终被释放为止，这是从空间上定义的，若从时间上可以称为生命周期。

5.3.1　全局变量与局部变量

Python 中的变量可根据作用范围，分为全局变量和局部变量两种。正确地定义和使用全局变量与局部变量，能够提高程序设计的效率。

说明：

(1) 全局变量在全部程序段中均有效，而局部变量只在定义的程序段以及下层程序段中才有效。

(2) 全局变量只能用显式操作才能清除，而局部变量在其程序段运行结束时将自动清除。

(3) 在主程序中定义的变量，将在全部程序段中有效，作用相当于全局变量。

(4) 在一个语句组内定义的局部变量能屏蔽同名的全局变量和高层程序段中的同名局部变量，当该程序段运行结束时，被屏蔽的全局变量将自动恢复。

(5) 全局变量通常是在函数定义的前进行赋值的，用于扩展变量在整个程序文件中的作用范围，请看下例。

【例 5-20】 全局变量示例。

源程序如下：

```
#定义全局变量
A=13
B=-8
#定义函数
def most(x,y):
    if x>y:
        z=x
    else:
        z=y
    return z
#主程序及其调用函数
print("A=",A,"\tB=",B)
print("较大数：",most(A,B))
```

本例中的主程序沿用前面定义的全局变量，并作为实参传递给被调用函数。

运行结果如图 5-19 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
A= 13	B= -8
较大数： 13
```

图 5-19　例 5-20 的运行结果

5.3.2　global 语句

定义全局变量还可以使用 global 语句，该语句是在函数内部进行变量定义的。尽管 Python 允许在函数前面定义全局变量，但还是建议最好避免这种用法，以便读者清楚一个变量的作用域。global 语句专门用于定义全局变量，若要使用一条 global 语句定义多个全局变量，则直接书写为 global a，b，c 即可。

【例 5-21】　global 语句使用示例。

源程序如下：

```
#第一次定义全局变量 a
a=6
#定义函数
def power(n):
    #第二次定义全局变量 a
    global a
    y=1
    for i in range(1,n+1,1) : y=y*a
    return y
#主程序及其调用函数
b=4
m=int(input("输入数据:"))
c=a*b;
print(a,"×",b,"=",c)
```

```
d=power(m);
print(a,"和",m,"乘方=",d)
```

本例中的函数中声明全局变量 a,从而能够引用并计算,其 a 值并没有进行参数传递。运行结果如图 5-20 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入数据:3
6 × 4 = 24
6 和 3 乘方= 216
```

图 5-20　例 5-21 的运行结果

【例 5-22】　求出 10 个整数中的最大数和最小数。

源程序如下:

```
#定义全局变量并进行初始化
most=99999
least=-9999
#定义函数
def two(b):
    global most                                   #声明全局变量
    global least                                  #声明全局变量
    most=b[0]
    least=b[0]
    for i in range(1,10):
        if most>b[i] : most=b[i]
        if least<b[i] : least=b[i]
#主程序及其调用函数
data=[1,2,3,4,5,0,9,8,7,6]
two(data)
print("最大数: ",most)
print("最小数: ",least)
```

本例中的函数并没有返回语句,而是在函数中利用全局变量 most 和 least,在其中生成最大数和最小数,然后让主程序引用。同时,伴随的结果是函数可以计算多个值,突破计算机函数中只能计算单值的限制。

运行结果如图 5-21 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
最大数:  0
最小数:  9
```

图 5-21　例 5-22 的运行结果

【例 5-23】　在一维列表内存放 10 个成绩数据,求平均值、最大值和最小值。

源程序如下:

```
#定义函数
#most=-9999
#least=99999
def aver(a,n):
```

```
    global most,least
    sum=a[0];
    most=least=a[0]
    for i in range(1,n,1):
        if a[i]>most : most=a[i]
        if a[i]<least : least=a[i]
        sum=sum+a[i]
    aver=sum/n;
    return (aver);
#主程序及其调用函数
score=[86,68,90,78,82,64,56,98,83,81,76,72]
av=aver(score,len(score))
print("最大值：",most,"\n最小值：",least,"\n平均值：",av)
```

本例中的 aver()函数只能返回平均值，而最大值和最小值是通过全局变量传递的。具体运行时序是，首先在程序开头定义并初始化全局变量，然后在 aver()函数中生成，最后在主程序中引用。最终结果是，通过 aver()函数可以计算出 3 个值。

运行结果如图 5-22 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
最大值：  98
最小值：  56
平均值：  77.83333333333333
```

图 5-22　例 5-23 的运行结果

5.3.3　变量同名

在出现全局变量与局部变量同名时，Python 规定局部变量优于全局变量。

【例 5-24】　全局变量与局部变量同名。

源程序如下：

```
a=3
b=5
print("a=",a)
#定义函数
def most(a,b):
    if a>b:
        return a
    else:
        return b
#主程序及其调用函数
a=8
print(a,b,"中的较大值",most(a,b))
```

本例中的主程序中再次定义局部变量 a 并初始化为 8，它优于第 1 行定义的全局变量 a，即实参就是数据 8，而不是 3。

运行结果如图 5-23 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
a= 3
8 5 中的较大值 8
```

图 5-23　例 5-24 的运行结果

5.4　匿名函数

匿名(即无名)函数在 Python 中具有唯一性,但使用上通常会与 lambda 函数相关。

5.4.1　lambda 函数

lambda 一词对应希腊字母 λ,与数学中的 λ 演算相关,该演算是用于研究函数定义、应用和递归的一套形式系统。在 Python 语言中,lambda 则是一个匿名函数,即没有函数名称的函数。lambda 直接由一个表达式来表示函数,显然比使用 def 语句定义函数简单,当然函数功能也必然简单,即只能用来编写一个运算式。两者比较,def 语句可以实现简单的 lambda 函数,而 lambda 则不能设计功能复杂的 def 语句。由于使用 lambda 定义函数时没有指定函数名,所以定义的只是关于一个表达式的匿名函数,其值将由 lambda 返回。

lambda 函数的一般格式如下:

```
lambda <参数 1>, <参数 2>, …, <参数 n>, <表达式>
```

说明:

<参数 1>,<参数 2>,…,<参数 *n*>:表达式中涉及的参数;

<表达式>:由以上参数构成的一个计算表达式。

在编程运用时,lambda 函数可以赋值给一个变量,或者作为列表常量,还能以参数形式出现在另外的函数调用中。

5.4.2　程序示例

【例 5-25】　lambda 表达式示例。

源程序如下:

```
#定义函数及其 lambda 表达式
def com(n):
    if(n==1):
        return lambda x,y:x+y
    if(n==2):
        return lambda x,y:x-y
#主程序及两次函数调用
print("输出调用结果: ")
operate=com(1)
print("46+2=",operate(46,2))
operate=com(2)
print("46-2=",operate(46,2))
```

本例中定义 com()函数及其 lambda 表达式,使用多分支语句分别定义两数的加、减运算,然后由 operate 对象实现具体的函数调用。

运行结果如图 5-24 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输出调用结果:
46+2= 48
46-2= 44
```

图 5-24　例 5-25 的运行结果

5.5　递归函数

在调用函数的过程中又出现直接或间接地调用该函数本身,称为函数的递归调用。Python 中允许函数进行递归调用,换句话说,可以直接调用函数自己,也可以间接调用函数自己。所谓间接调用是指函数 A 中调用函数 B,而函数 B 中又调用函数 A。本节只介绍直接递归。

来看如下关于一个整数的函数定义:

$f(n)= n\times f(n-1)$,且初值 $f(1)$ 为 1。

如要计算 $f(5)$,则有

$f(5)= 5\times f(4)= 5\times 4\times f(3)=20\times 3\times f(2)=60\times 2\times f(1)=120\times 1$,即 5!。

在计算机语言(例如 Java、C、Python 等)中均引入递归函数,这是因为递归函数具有两个好处,一是编程方便,可用简单分支结构代替复杂的循环结构;二是用于求解特殊问题,这些问题若不用递归函数是极难解决的。

5.5.1　递归函数及其调用

一个计算问题要采用递归调用时,必须符合以下 3 个条件:

(1) 有一个明确的结束递归调用过程的条件。

(2) 可以运用转化过程使问题得以简化并加以解决。

(3) 可以将所求解的计算问题转化为另一个简化的计算问题,而二者之间的解法是完全相同的,被处理的对象必须有规律地递增或递减。

5.5.2　程序示例

【例 5-26】　用递归方法求 n!

求解方法:这里没有使用循环结构,而是使用递归方法编写如下的二分段函数:

$$f(n)=\begin{cases}1, & n=1\\ n\times f(n-1), & n \text{ 为大于 1 的整数}\end{cases}$$

源程序如下:

```
#定义递归函数
def fact(n):
    if n==1:                                        #递归调用结束的条件
        return 1
```

```
    else:
        return fact(n-1) * n                        #递归调用
#主程序并调用递归函数
n=int(input("输入数据："))
print(n,"的阶乘：",fact(n))
```

运行结果如图 5-25 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入数据：10
10 的阶乘： 3628800
```

图 5-25　例 5-36 的运行结果

从本例中可以发现，递归函数通常是分段函数，程序中使用 if 语句就能够实现，编程非常简单，并没有使用循环语句，但是机器在实现递归函数时，只能通过重复调用参数不同的同一函数而最终实现，这本身就是循环。当然，递归过程将增加机器负担，但可以提高编程效率。

【例 5-27】 求出斐波那契数列 1，2，3，5，8，13，…的第 10 个数。

求解方法：写出斐波那契数列的递归函数如下：

$$f(n)=\begin{cases}1, & n=1\\ 2, & n=2\\ f(n-1)+f(n-2), & n\text{ 为大于 2 的整数}\end{cases}$$

源程序如下：

```
#定义递归函数
def fib(n):
    if n==1:                                    #调用结束的条件
        return 1
    elif n==2:                                  #调用结束的条件
        return 2
    else:
        return fib(n-2)+fib(n-1)                #递归调用
#主程序及其调用递归函数
n=int(input("输入数据："))
print("第",n,"个斐波那契数：",fib(n))
```

运行结果如图 5-26 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入数据：10
第 10 个斐波那契数： 89
```

图 5-26　例 5-27 的运行结果

本例中的第 2～9 行定义递归函数 $f()$，其中使用多分支 if 语句构成斐波那契数列，并没有使用循环语句。只不过递归调用过程的机器实现相当于循环，这样能够提高编程效率。例如求斐波那契数列中第 5 个数的递归调用过程如下：

$f(5)=f(4)+f(3)=f(3)+f(2)+f(2)+f(1)=f(2)+f(1)+f(2)+f(2)+f(1)=2+1+2+2+1=8$。

【例 5-28】 有 5 个人坐在一起,问第 5 个人多少岁,他说比第 4 个人大 2 岁。问第 4 个人,他说比第 3 个人大 2 岁。问第 3 个人,又说比第 2 个人大 2 岁。问第 2 个人,说比第一个人大 2 岁。最后问第一个人,他说是 10 岁。请问第 5 个人多少岁。

求解方法:写出递归函数如下:

$$\mathrm{age}(n)=\begin{cases}10, & n=1\\ \mathrm{age}(n-1)+2, & n\ \text{为大于 1 的整数}\end{cases}$$

源程序如下:

```
#定义递归函数
def age(n):
    if n==1:                                    #递归调用结束的条件
        a=10
    else:
        a=age(n-1)+2                            #递归调用
    return a
#主函数及其调用递归函数
print("第 5 个人的岁数是",age(5))
```

递归函数 age()使用多分支 if 语句构成计算过程,具体情况如下:

age(5)=2+age(4)=4+age(3)=6+age(2)=8+age(1)=18。

运行结果:

```
第 5 个人的岁数是 18
```

【例 5-29】 求整数 1~10 的平方和。

写出递归函数如下:

$$s(n)=\begin{cases}1, & n=1\\ s(n-1)+n\cdot n, & n\ \text{为大于 1 的整数}\end{cases}$$

源程序如下:

```
#定义递归函数
def sum(x):
   if x==1:                                     #递归调用结束的条件
      return 1
   else:
      return(sum(x-1)+x*x)                      #递归调用
#主函数及其调用递归函数
n=int(input("输入数据: "))
print("计算结果: ",sum(n))
```

运行结果如图 5-27 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入数据: 10
计算结果:  385
```

图 5-27 例 5-29 的运行结果

【例 5-30】 模拟求解汉诺塔问题。问题大意是，设定 3 根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着 64 个圆盘，操作要求把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在 3 根柱子之间一次只能移动一个圆盘。

求解方法：设定 3 根柱子分别为 A、B、C，写出递归操作过程如下：

(1) 把前 $n-1$ 个盘子由 A 移到 B；

(2) 把第 n 个盘子由 A 移到 C；

(3) 把前 $n-1$ 个盘子由 B 移到 C。

源程序如下：

```
#定义一般函数 move()
def move(one,three):
    print(one,"→",three)
#定义递归函数 hanoi()
def hanoi(n,one,two,three):
    if (n==1):                                      #递归调用结束的条件
      move(one,three)
    else:
      hanoi(n-1,one,three,two)                      #将前 n-1 个盘子搬到中间柱
      move(one,three)                               #将最后一个盘子搬到目标柱
      hanoi(n-1,two,one,three)                      #将前 n-1 个盘子搬到目标柱
#主程序及其调用递归函数
n=int(input("圆盘数："))
print("移动",n,"个圆盘的过程：")
hanoi(n,"A","B","C")
```

运行结果如图 5-28 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
圆盘数：3
移动 3 个圆盘的过程：
A → C
A → B
C → B
A → C
B → A
B → C
A → C
```

图 5-28　例 5-30 的运行结果

汉诺塔问题的计算复杂度非常高，本例中设定只有 3 个圆盘。读者可以尝试 4 个及以上圆盘的递归操作过程。

【例 5-31】 使用递归函数计算正整数 x 和 y 的最大公约数。

求解方法：写出递归函数如下：

$$\gcd(x,y)=\begin{cases}x, & y=0\\ \gcd(y,x\%y), & y\text{ 为大于等于 0 的整数}\end{cases}$$

源程序如下：

```
#定义递归函数计算 x 和 y 的最大公约数
def gcd(x,y):
```

```
    if y==0:                          #递归调用结束的条件
        return x                      #如果 y 为 0 则返回 x
    else:
        return gcd(y,x%y)             #如果 y 大于 0 则递归调用 gcd(y, x%y)
#主程序及其调用递归函数
a=int(input("a="))
b=int(input("b="))
print(a,"和",b,"的最大公约数为",gcd(a,b))
```

运行结果如图 5-29 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
a=28
b=16
28 和 16 的最大公约数为 4
```

图 5-29　例 5-31 的运行结果

【例 5-32】 递归计算 n 阶调和数 $1+1/2+1/3+\cdots+1/n$。

求解方法：写出递归函数如下：

$$s(n)=\begin{cases}1, & n=1\\ \text{age}(n-1)+1/n, & n\text{ 为大于 1 的整数}\end{cases}$$

源程序如下：

```
#定义递归函数
def s(n):
    if n==1:                          #递归调用结束的条件
        return 1
    else:
        return s(n-1)+1/n             #递归调用
#主程序及其调用递归函数
n=int(input("输入数据："))
print("计算过程：")
#输出 1~n 阶的调和数
for i in range(1,n+1,1):
    print("第",i,"次：",s(i))
```

运行结果如图 5-30 所示。

```
====================== RESTART: D:/Python36/ch5/p1.py ======================
输入数据：6
计算过程：
第 1 次：  1
第 2 次：  1.5
第 3 次：  1.8333333333333333
第 4 次：  2.083333333333333
第 5 次：  2.283333333333333
第 6 次：  2.4499999999999997
```

图 5-30　例 5-32 的运行结果

【例 5-33】 有一只猴子第 1 天摘下若干个桃，当即吃掉一半，又多吃一个；第 2 天又将剩下的桃吃掉一半，又多吃一个；按照这样的吃法，每天都吃掉前一天剩下的一半又多吃一个。到第 10 天，就剩下一个桃。问这个猴子第一天摘了多少个桃。

求解方法：猴子一共吃 9 次，第 9 次吃过后，剩下一个桃子。设定第 n 次吃过的桃子数

为 c(n),则有递归函数如下:

$$c(n)=\begin{cases}1, & n=9\\ 2\times(c(n+1)+1), & n\text{ 为小于 9 的自然数}\end{cases}$$

注意:本例中的递归函数是升序递归到 9 时结束,前面的许多递归函数是降序递归到 1 时结束的。

源程序如下:

```
#定义递归函数
def c(n):
    if n==9:                          #递归调用结束的条件
        return 1
    else:
        return 2*(c(n+1)+1)
#主程序并调用函数
n=int(input("设置开始:"))
print("桃子总数:",c(0))
```

运行结果如图 5-31 所示。

```
======================= RESTART: D:/Python36/ch5/p1.py =======================
设置开始: 0
桃子总数:  1534
```

图 5-31　例 5-33 的运行结果

【例 5-34】　将一个正整数 n 转换成对应的反序数,例如输入 1234 则输出 4321。其中,正整数 n 的位数没有固定,可以是任意位数的整数。

求解方法:写出递归函数如下:

$$c(n)=\begin{cases}n, & 0<n<10\text{ 且 }n\text{ 为整数}\\ 10\times(n\%10+c(n//10)), & n\text{ 内大于 9 的整数}\end{cases}$$

源程序如下:

```
#定义递归函数
def convert(n):
    print(n%10,end="")
    temp=n//10
    if n//10!=0 : convert(n//10)
#主程序及其调用递归函数
number=int(input("原数据:"))
print("反序数:",end="")
convert(number)
```

本例中的 convert()函数并没有返回值,而是在函数体直接显示相应的数位,故主函数在调用函数时并没有变量对函数返回值进行引用。

运行结果如图 5-32 所示。

```
===================== RESTART: D:/Python36/ch5/p1.py =====================
原数据:1234567
反序数: 7654321
```

图 5-32　例 5-34 的运行结果

习　题　5

一、简答题

1. 什么是函数？计算机中为什么要使用函数？
2. 简述 Python 语言中的函数分类情况。
3. 简述函数定义与调用的方法。
4. 什么是函数的嵌套调用？有何作用？
5. 什么是形式参数？什么是实在参数？
6. 简述 Python 语言中的可变参数及其作用。
7. 什么是变量的作用域？
8. 什么是全局变量？什么是局部变量？
9. 简述 Python 语言中的 global 语句及其作用。
10. 什么是匿名函数？如何使用 lambda 表达式？
11. 什么是递归函数？如何定义和使用递归函数。

二、编程题

1. 编写函数计算 1－3＋5－7＋9－…＋97 的值并调用。
2. 编写函数找出正整数 n 的全部因数。
3. 编写函数实现交叉合并字符串，例如 ABCD 与 12345 的合并结果是 A1B2C3D45。
4. 编写函数递归函数求斐波那契数列的第 30 个数。
5. 编写函数求出两个正整数的最小公倍数。
6. 编写函数求解方程 $ax^2+bx+c=0$ 并调用，要求分别处理根的判别式小于 0、等于 0 和大于 0 的情况。
7. 编写函数将纯小数 d 转换为二进制形式并调用。（提示：使用乘二取整算法，并用列表存放全部二进制的数位。例如输入 0.12345，则输出 0.0001111110011010011 0。）
8. 利用可变参数计算 10 以内的全部阶乘。

第6章　模　　块

模块是计算思维的重要概念，也是组织程序(尤其大型程序)的一种高级形式，用于实现对程序和相关数据的封装，为程序员调用模块中的函数和常量提供方便。Python 语言将任何程序文件都视为模块，以使程序规模构建到更大程度。本章介绍 math、cmath、decimal、fractions、random、time、datetime、calendar、time、os、sys 等 Python 标准库中的内置模块。最后，还介绍如何自定义模块和包。像 NumPy、SciPy 等需要另行安装后才能调用的第三方模块本章不会介绍。

6.1　模　　块

在 Python 编程规范中，程序结构通常是先导入指定的模块，后调用模块中的函数和常量(或称属性)。

6.1.1　导入模块

Python 中可以使用许多模块，但需要调用前必须导入指定模块，最常用的模块导入方法是使用 import 语句。使用 import 语句可以实现对模块的整体导入，其一般引用格式如下：

格式 1：

```
import <模块名>
```

格式 2：

```
import <模块名>as <模块别名>
```

格式 3：

```
import <模块名 1>, <模块名 2>, …, <模块名 n>
```

这里的＜模块名＞与＜模块别名＞均是区分大小写的，最好不要一次导入过多的模块。

在使用 import 导入模块后，调用模块中的函数和常量均需要用圆点运算符，具体形式和使用内置数据类型的方法相同，即：

```
<模块名>.<变量名>
```

或

```
<别名>.<变量名>
```

注意：用 import 语句导入模块后，调用时函数和常量前均必须加模块名或别名。如果没有使用模块名或别名，系统将触发 NameError 异常。

【例 6-1】 导入 math 模块并调用函数。

在 IDLE 交互环境中,输入如下命令:

```
>>>import math          #导入 math 模块
>>>math.sqrt(5)         #正常调用函数
>>>sqrt(5)              #非正常调用函数将触发异常
```

运行结果如图 6-1 所示。

```
>>> import math
>>> math.sqrt(5)
2.23606797749979
>>> sqrt(5)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    sqrt(5)
NameError: name 'sqrt' is not defined
```

图 6-1 例 6-1 的运行结果

从运行结果中可以发现,在调用 sqrt()函数前若没有加模块名,则系统将触发 NameError 异常,表示系统不能识别 sqrt()函数,实质是不知道必须在 math 模块中调用函数。

【例 6-2】 导入 math 模块并实现函数和常量的调用。

源程序如下:

```
#导入 math 模块
import math
print("pi=\t\t",math.pi)
print("e=\t\t",math.e)
print("sqrt(2)=\t",math.sqrt(2))
```

本例中的第 2 行使用 import 语句导入 math 模块,其后才能实现对两个常量和一个函数的调用。

运行结果如图 6-2 所示。

```
======================== RESTART: D:/Python36/ch6/p1.py ========================
pi=                     3.141592653589793
e=                      2.718281828459045
sqrt(2)=                1.4142135623730951
```

图 6-2 例 6-2 的运行结果

6.1.2 导入模块成员

若仅导入模块中的指定成员,则应该使用 from…import 语句。

(1) 导入模块中的部分成员,可使用如下格式:

```
from <模块名>import <成员 1>, <成员 2>, …, <成员 n>
```

(2) 导入模块中的一个成员,可使用如下格式:

```
from <模块名>import <成员>
```

由于 Python 具有大量的模块以及开源本质,所以在导入模块时最好按照如下顺序:Python 标准库模块、Python 第三方模块和自定义模块。

【例 6-3】 导入 math 模块中的一个成员函数并进行调用,以便计算 9 个正弦值。

源程序如下:

```
from math import sin
print("显示正弦值:")
for d in range(10,91,10):
    print(d,"度:\t",sin(d*3.14/180))
```

本例中的第1行使用 from…import 语句仅导入 math 模块中的一个成员函数 sin(),其后才能进行函数调用。要注意的是,在这种情况下书写函数调用时不能添加前缀 math,否则将出现 NameError 异常。

运行结果如图6-3所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
显示正弦值:
10 度:	0.17356104045380674
20 度:	0.34185384854620343
30 度:	0.49977010264310240
40 度:	0.6425164486712008
50 度:	0.7657599964977133
60 度:	0.8657598394923444
70 度:	0.9394806051566189
80 度:	0.9846845901305833
90 度:	0.9999996829318346
```

图6-3　例6-3的运行结果

【例6-4】 导入 math 模块中的3个成员函数并进行调用。

源程序如下:

```
from math import fabs,sqrt,pow
print("fabs(-3):\t",fabs(-3))          #实数取绝对值
print("sqrt(3):\t",sqrt(3))            #实数开平方根
print("pow(2,5):\t",pow(2,5))          #实数进行指数运算
```

本例中的第1行使用 from…import 语句导入 math 模块中的3个成员函数,其后才能进行相应的函数调用。

运行结果如图6-4所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
fabs(-3):	3.0
sqrt(3):	1.7320508075688772
pow(2,5):	32.0
```

图6-4　例6-4的运行结果

综上所述,import 和 from 导入模块的效果各异,使用何条语句来导入模块应该与应用需求相关。通常可以用 import 导入简单模块,在可能出现变量名称冲突时使用 from 指定导入模块中的相关函数或常量。

当然,若要访问模块中的常量,也可以使用 from…import 语句,这里不再举例。

6.1.3 模块搜索路径

众所周知,任何文件系统中的文件数量(多达数万)都非常大,这使得合理的搜索路径至关重要,下面说明在 IDLE 交互环境下的四类路径及其使用。

1. Python 安装文件夹

这就是程序的主文件夹,是模块搜索的最优先位置。例如,若 Python 系统安装在 D:\

Python36 文件夹中，则 Python 解释器将首先在该文件夹中搜索模块是否存在。

2. 环境变量 PYTHONPATH

PYTHONPATH 是可以由用户自行定义的搜索文件夹的变量。也就是说，如果已知要导入模块在某个文件夹下，只要将这个位置添加到环境变量中就可以让解释器顺利搜索到该模块是否导入。

【例 6-5】 在命令提示符窗口中设置 PYTHONPATH 变量。

要将一个模块路径添加到 PYTHONPATH 变量中，可以在命令提示符窗口中输入命令：

```
D:\>set PythonPATH=D:\Python36\my_modules
```

操作过程如图 6-5 所示。

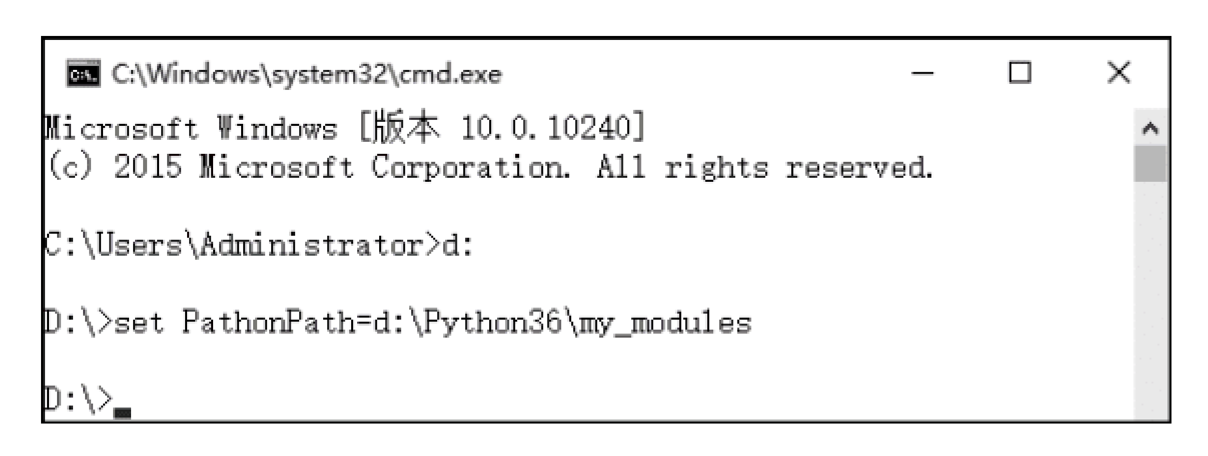


图 6-5 例 6-5 的运行结果

在命令提示符窗口中，Windows 系统响应命令的形式有如下 3 种。

(1) 命令完全正确：机器正常执行命令，并没有其他显示，如例 6-5 所示。

(2) 命令书写错误：机器显示相关命令的出错信息。

(3) 命令书写正确但逻辑错误：机器执行后将产生副作用。

3. 标准库文件夹

Python 在安装时，默认将全部内置模块放在到安装路径下的 Lib 文件夹中。

4. 路径文件.pth

路径文件.pth 是用户自行建立的文本文件，以设置所需的模块搜索路径。若系统存在.pth 文件，则系统将搜索在该文件中所列出的路径。

再重申一次，为了问题描述简洁、统一，本书将 Python 系统安装在 D:\Python36 中。

6.2 数值类模块

程序中经常要运用到数学和工程方面的计算，这时就可以调用 math 模块和 cmath 模块。其中，math 模块实现实数的数学计算，cmath 模块实现复数的数学计算。

6.2.1 math 模块

1. 查看函数列表

要查看 math 模块中的全部函数名，可在 IDLE 交互环境中输入如下命令：

```
>>>import math
>>>dir(math)
```

这样就可以得到 math 模块中的全部函数名列表,如图 6-6 所示。

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__'
, 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', '
ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc'
, 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp'
, 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite'
, 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

图 6-6 math 模块中的全部函数名列表

2. 常用函数

正如图 6-6 所示,math 模块中的函数非常多,常用的函数如表 6-1 所示。

表 6-1 常用函数

函数	说明	函数	说明
ceil(x)	返回大于或等于 x 的最小整数	floor(x)	返回小于或等于 x 的最大整数
fabs(x)	返回绝对值	factorial(x)	返回 x 的阶乘的值
hypot(x,y)	返回直角三角形的斜边长的值	pow(x,y)	返回 x 的 y 次方的值
sqrt(x)	返回正数 x 的平方根的值	log(x)	返回 x 的自然对数的值
log10(x)	返回 x 的常用对数	trunc(x)	返回截尾取整结果
isnan(x)	判断是否 NaN	degrees(x)	弧度转角度
radians(x)	角度转弧度		

注意:常量 NaN 的含义是 not a number,表示不是数值。

3. 程序示例

【例 6-6】 math 模块中的函数调用示例。

源程序如下:

```
import math
print("ceil(2.56)=\t",math.ceil(2.56))
print("floor(2.56)=\t",math.floor(2.56))
print("factorial(5)=\t",math.factorial(5))
print("hypot(3,4)=\t",math.hypot(3,4))
print("pow(3,4)=\t",math.pow(3,4))
print("sqrt(3)=\t",math.sqrt(3))
print("log(3)=\t\t",math.log(3))
print("log10(3)=\t",math.log10(3))
print("trunc(2.56)=\t",math.trunc(2.56))
print("isnan(3)=\t",math.isnan(3))
print("degrees(3)=\t",math.degrees(3))
print("radians(360)=\t",math.radians(360))
```

运行结果如图 6-7 所示。

从运行结果可以发现,绝大多数函数的计算结果都按统一方式获得实数,例如 4 的平方

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
ceil(2.56)=        3
floor(2.56)=       2
factorial(5)=      120
hypot(3,4)=        5.0
pow(3,4)=          81.0
sqrt(3)=           1.7320508075688772
log(3)=            1.0986122886681098
log10(3)=          0.47712125471966244
trunc(2.56)=       2
isnan(3)=          False
degrees(3)=        171.88733853924697
radians(360)=      6.283185307179586
```

图 6-7　例 6-6 的运行结果

根为实数 2.0，而不是整数 2。

6.2.2　cmath 模块

cmath 模块包含一些关于复数运算的函数，这些函数与 math 模块中的函数基本保持一致，区别在于 cmath 模块运算的数据是复数。

【例 6-7】　cmath 模块中的函数示例。

源程序如下：

```
#导入 cmath 模块
import cmath
print("cmath.sqrt(-2)=\t",cmath.sqrt(-2))
print("cmath.sqrt(2)=\t",cmath.sqrt(2))
print("cmath.sin(2)=\t",cmath.sin(2))
print("log10(20)=\t",cmath.log10(20))
```

运行结果如图 6-8 所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
cmath.sqrt(-2)=   1.4142135623730951j
cmath.sqrt(2)=    (1.4142135623730951+0j)
cmath.sin(2)=     (0.9092974268256817-0j)
log10(20)=        (1.3010299956639812+0j)
```

图 6-8　例 6-7 的运行结果

6.2.3　decimal 模块

decimal 模块专门用于十进制浮点数运算，以便实现对工程与科学方面的计算支持。decimal 模块具有如下特点：

（1）提供十进制的数据类型，并且以十进制数形式进行存储；

（2）存储固定位数的数字，以便由程序指定精度；

（3）十进制数据中的小数点位置是可变的，从而与规格化的实型数据区别开来。

1. 十进制浮点数运算

decimal 模块提供 Decimal 类型专门用于浮点数计算。与内置的二进制浮点数实现的 float 类型相比较，Decimal 类型有助于金融应用和其他需要精确十进制表示的场合。在控制精度、舍入误差等方面更能适应规定要求，确保十进制数位的精度，或者满足用户所希望的计算结果与人工计算保持一致。

2. 程序示例

十进制浮点数运算的结果总是保有结尾的 0，自动从两位精度延伸到 6 位。Decimal 重现人工方式的数学运算，这就确保由二进制浮点数无法精确保证的数据精度。

【例 6-8】 计算 0.7×1.05 的两种方法。

源程序如下：

```
#导入 decimal 模块
from decimal import *
dgt=Decimal("0.70")*Decimal("1.05")                #十进制浮点数运算
print("Decimal 数据计算：\t",round(dgt,2))
print("Float 数据计算：\t\t",round(0.7*1.05,2))     #实型数据运算
```

运行结果如图 6-9 所示。

```
======================= RESTART: D:/Python36/ch6/p1.py =======================
Decimal数据计算:            0.74
Float数据计算:              0.73
```

图 6-9　例 6-8 的运行结果

从运行结果可以发现，Decimal 数据具有更高的计算精度，这是源于 Decimal 数据没有十进制数与二进制数之间相互转换导致的误差。

【例 6-9】 Decimal 数据的取模运算。

源程序如下：

```
from decimal import *
print(Decimal("1.00")%Decimal("0.10"))
print(1.00%0.10)
```

本例中的第 2 行使用 Decimal 数据运算，而第 3 行使用 Float 数据运算。从运行结果中可以发现，高精度的 Decimal 数据能够执行二进制浮点数无法进行的取余运算。

运行结果如图 6-10 所示。

```
======================= RESTART: D:/Python36/ch6/p1.py =======================
0.00
0.09999999999999995
```

图 6-10　例 6-9 的运行结果

【例 6-10】 Decimal 数据实现等值测试。

在 IDLE 交互环境中，输入如下命令：

```
>>>from decimal import *
>>>sum([Decimal("0.1")]*10)==Decimal("1.0")
>>>sum([0.1]*10)==1.0
```

运行结果如图 6-11 所示。

```
>>> from decimal import *
>>> sum([Decimal("0.1")]*10)==Decimal("1.0")
True
>>> sum([0.1]*10)==1.0
False
```

图 6-11　例 6-10 的运行结果

从运行结果中可以发现,Decimal 数据能够执行二进制浮点数无法进行的等值测试。

【例 6-11】 Decimal 数据实现高精度计算。

源程序如下:

```
from decimal import *
getcontext().prec=32
print("精度 32 位: \n\t",Decimal(2)/Decimal(7))
```

在本例的第 2 行中,设置浮点数的精度是 32 位,第 3 行调用 Decimal()函数强制将两个整数转换成 Decimal 类型并相除,从而得到 32 位精度的结果。这里的 getcontext()函数实现浮点数精度的设置,而设置为 32 位精度是绝大多数语言无法达到的。

运行结果如图 6-12 所示。

```
======================= RESTART: D:/Python36/ch6/p1.py =======================
精度32位:
	 0.28571428571428571428571428571429
```

图 6-12　例 6-11 的运行结果

6.2.4　fractions 模块

1. fractions 模块

fractions 模块用于处理分数类型的数据,与其他数值类型相同的是,全部运算需要两个数据,即二元运算。具体运算效果可以分为 3 种:

(1) 两个分数相加得到一个分数,即使两个 1/3 相加,结果也是 2/3;

(2) 一个分数加一个整数得到一个分数;

(3) 一个分数加一个实数得到一个实数。

实际上,后两种运算效果相当于数据类型转换,即浮点数优于分数,分数优于整数。

2. 程序示例

【例 6-12】 混合二元运算示例。

源程序如下:

```
#导入 fractions 模块
from fractions import Fraction
#两个分数相加
print("Fraction(1,2)+Fraction(1,2):\t",Fraction(1,2)+Fraction(1,2))
#分数与整数相加
print("Fraction(1,2)+1:\t\t",Fraction(1,2)+1)
#分数与实数相加
print("Fraction(1,2)+1.5:\t\t",Fraction(1,2)+1.5)
```

运行结果如图 6-13 所示。

```
======================= RESTART: D:/Python36/ch6/p1.py =======================
Fraction(1,2)+Fraction(1,2):	 1
Fraction(1,2)+1:		 3/2
Fraction(1,2)+1.5:		 2.0
```

图 6-13　例 6-12 的运行结果

【例 6-13】 找出两个正整数的最大公约数。

在 IDLE 交互环境中，输入如下命令：

```
>>>from fractions import gcd
>>>gcd(28, 16)
```

本例中的 gcd()函数用于找出两个正整数的最大公约数。

运行结果：4。

【例 6-14】 化简分式。

在 IDLE 交互环境中，输入如下命令：

```
>>>from fractions import Fraction
>>>Fraction(1, 2)+Fraction(1, 3)+Fraction(1, 6)
>>>print(Fraction(1, 2)+Fraction(1, 3)+Fraction(1, 6))
```

运行结果如图 6-14 所示。

```
>>> from fractions import Fraction
>>> Fraction(1,2)+Fraction(1,3)+Fraction(1,6)
Fraction(1, 1)
>>> print(Fraction(1,2)+Fraction(1,3)+Fraction(1,6))
1
```

图 6-14 例 6-14 的运行结果

从运行结果可以发现，fractions 模块具有化简分式的功能，能够实现约分和通分。另外，注意数据类型的一致性，即分数 1/1 不是整数 1。实际上，数据类型在绝大多数情况下是优先于数值的，即计算机通常是由数据类型而不是数值来确定运算方式的。

【例 6-15】 编程描述计算圆周率的 3 种分数形式：

(1) 3＋1/7

(2) (3＋1/(7＋1/15))

(3) (3＋1/(7＋1/(15＋1/25)))

求解方法：这 3 种分数是逐次展开的，其运算表达式是由括号才能界定的。

源程序如下：

```
from fractions import Fraction
print("第 1 次取分数：",end="\t")
print(3+Fraction(1,7))
print("第 2 次取分数：",end="\t")
print(3+Fraction(1,7+Fraction(1,15)))
print("第 3 次取分数：",end="\t")
print(3+Fraction(1,7+Fraction(1,15+Fraction(1,25))))
```

本例中的第 1 行是由＃导入实现初始化操作的构造函数 Fraction()，实现将两个整数转换成分数，例如 Fraction(1,7)将得到 1/7。

运行结果如图 6-15 所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
第1次取分数:	22/7
第2次取分数:	333/106
第3次取分数:	8347/2657
```

图 6-15 例 6-15 的运行结果

6.3　random 模块

random 模块用于生成随机数，下面介绍 random 模块中的主要应用。

6.3.1　常用函数

random 模块中的常用函数如表 6-2 所示。

表 6-2　常用函数

函　　数	说　　明
choice(＜sequence＞)	＜sequence＞表示序列，从中获取随机数
randint(a,b)	随机生成一个范围为 a～b 的整数，参数表示上限和下限
random()	随机生成一个范围为 0～1 的浮点数
randrange([start],stop[,step])	在指定范围内按指定基数递增集合中获取随机数
sample(＜sequence＞,k)	从指定序列中随机获取指定长度的片断
shuffle(x[,＜random]＞)	将指定列表中的元素打乱
uniform(a,b)	随机生成一个范围为 a～b 的浮点数，参数表示上限和下限

在调用以上函数前，必须导入 random 模块，其引用格式如下：

```
>>>import random
```

【例 6-16】　随机生成范围为 0～1000 的整数。

在 IDLE 交互环境中，输入如下命令：

```
>>>import random
>>>random.randint(0, 1001)
```

本例中两次重复调用 randint()函数将得到两个不同的随机数。

运行结果如图 6-16 所示。

```
>>> import random
>>> random.randint(0,1001)
921
>>> random.randint(0,1001)
968
```

图 6-16　例 6-16 的运行结果

【例 6-17】　随机生成范围为 0～1000 的偶数。

在 IDLE 交互环境中，输入如下命令：

```
>>>random.randrange(0, 1001, 2)
```

本例中两次重复调用 random()函数将得到两个不同的随机数。

运行结果如图 6-17 所示。

【例 6-18】　生成有效位数为 18 位的随机浮点数。

在 IDLE 交互环境中，输入如下命令：

```
>>> import random
>>> random.randrange(0,1001,2)
962
>>> random.randrange(0,1001,2)
342
```

图 6-17　例 6-17 的运行结果

```
>>>random.random()
```

运行结果如图 6-18 所示。

```
>>> import random
>>> random.random()
0.9418404821439554
>>> random.random()
0.9831323534110188
```

图 6-18　例 6-18 的运行结果

【例 6-19】 从指定字符串中生成随机字符。

在 IDLE 交互环境中,输入如下命令:

```
>>>random.choice("abcd12345-=")
```

本例中两次重复调用 choice()函数将得到 2 个不同的随机数。

运行结果如图 6-19 所示。

```
>>> import random
>>> random.choice("abcd12345-=")
'b'
>>> random.choice("abcd12345-=")
'4'
```

图 6-19　例 6-19 的运行结果

【例 6-20】 随机选取字符串。

在 IDLE 交互环境中,输入如下命令:

```
>>>random.choice(["red", "green", "blue", "black", "white", "gray"])
```

运行结果如图 6-20 所示。

```
>>> import random
>>> random.choice(["red","green","blue","black","white","gray"])
'black'
>>> random.choice(["red","green","blue","black","white","gray"])
'gray'
```

图 6-20　例 6-20 的运行结果

【例 6-21】 从指定字符串中的选取特定数量的字符。

在 IDLE 交互环境中,输入如下命令:

```
>>>random.sample("12345678909876543210", 5)
```

运行结果如图 6-21 所示。

```
>>> import random
>>> random.sample("12345678909876543210",5)
['3', '5', '6', '3', '1']
>>> random.sample("12345678909876543210",5)
['2', '9', '8', '2', '1']
```

图 6-21　例 6-21 的运行结果

在本例中，两次命令执行后可能获得相同的结果，这只是偶然发生的情况。这是因为在数据集合太小时，随机数的生成效果不够好。

【例 6-22】 随机改变列表中的元素排列。

在 IDLE 交互环境中，输入如下命令：

```
>>>items=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
>>>random.shuffle(items)
>>>items
```

本例中的列表 items 初始化为有序的 13 个数，通过调用函数 shuffle()使列表元素重新排列。

运行结果如图 6-22 所示。

```
>>> import random
>>> items=[1,2,3,4,5,6,7,8,9,10,11,12,13]
>>> random.shuffle(items)
>>> items
[4, 11, 1, 8, 9, 6, 5, 10, 7, 3, 2, 13, 12]
```

图 6-22 例 6-22 的运行结果

6.3.2 程序示例

【例 6-23】 实现两位正整数的求和测试。

求解方法：调用 random 模块中的随机函数生成两个 10～99 的正整数，分别由变量 a 和 b 表示，要求用户输入求和结果，由程序测试是否正确。

源程序如下：

```
import random
a=random.randint(10,100)
b=random.randint(10,100)
print("计算",a,"+",b,end="=\t")
s=int(input())
if s==a+b:
    print("求和正确")
else:
    print("求和错误")
```

运行结果如图 6-23 所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
计算 41 + 72=    123
求和错误
>>>
====================== RESTART: D:/Python36/ch6/p1.py ======================
计算 63 + 47=    110
求和正确
```

图 6-23 例 6-23 的运行结果

以上程序共执行两次，一次测试正确，一次测试失败。

【例 6-24】 将 1～52 这 52 个整数随机排列。

求解方法：若设定扑克牌中的花色大小顺序为梅花→方块→红桃→黑桃，点数大小顺

序为 2→3→…→10→J→Q→K→A，则全副扑克牌（没有大小王）可以由 1～52 的整数来表示，从而洗牌就是将 1～52 的整数重新排列，当然要分别均分成 4 手，每手 13 张扑克牌。

源程序如下：

```
import random
items=[]
#生成列表并初始化为 1~52 的有序整数
for i in range(1,53,1):
    items.append(i)
print("原始数据：")
#显示原始的列表
for n in range(0,4,1):
    for k in range(0,13,1):
        print(items[n*13+k],end=", ")
    print()
#混洗 4 手，每手 13 张扑克牌
random.shuffle(items)
#显示混洗的列表
print("随机数据：")
for n in range(0,4,1):
    for k in range(0,13,1):
        print(items[n*13+k],end=", ")
    print()
```

运行结果如图 6-24 所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
原始数据:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
随机数据:
14, 32, 8, 46, 50, 52, 51, 41, 29, 5, 35, 6, 12,
16, 20, 36, 11, 37, 44, 33, 42, 47, 17, 26, 30, 38,
43, 49, 7, 28, 22, 34, 24, 31, 19, 40, 9, 13, 10,
21, 18, 25, 15, 1, 39, 48, 23, 2, 45, 3, 27, 4,
```

图 6-24　例 6-24 的运行结果

【例 6-25】 洗牌程序。

random 模块中提供实现洗牌功能的 shuffle()函数，以便扑克爱好者直接在网络环境中玩扑克牌游戏，例如桥牌竞技。注意，shuffle()函数只实现 52 张扑克牌的混洗，其中没有大小王。

求解方法：调用 random 模块中的 shuffle()函数就可以实现洗牌，并用 4 个列表保存 4 手，每手 13 张扑克牌。

源程序如下：

```
import random
#Club(梅花)、Diamond(方块)、Heart(红桃)、Spade(黑桃)、2-10, J, Q, K, A
#用列表表示一副扑克牌并初始化，例如字符串 C7 表示梅花 7
cards=["C2","C3","C4","C5","C6","C7","C8","C9","C10","CJ","CQ","CK","CA",
```

```
        "D2","D3","D4","D5","D6","D7","D8","D9","D10","DJ","DQ","DK","DA",
        "H2","H3","H4","H5","H6","H7","H8","H9","H10","HJ","HQ","HK","HA",
        "S2","H3","S4","S5","S6","S7","S8","S9","S10","SJ","SQ","SK","SA"]
#调用 shuffle()函数实现洗牌，即列表 cards 中的元素混排
random.shuffle(cards)
#初始化 4 手牌
pack1=[]
pack2=[]
pack3=[]
pack4=[]
#循环 13 次，每次将列表 cards 中的 4 个元素添加至 pack1、pack2、pack3 和 pack4 中
for i in range(0,13,1):
    #从列表 cards 中取首部的 4 个元素分别添加至 pack1、pack2、pack3 和 pack4 中
    pack1.append(cards.pop())
    pack2.append(cards.pop())
    pack3.append(cards.pop())
    pack4.append(cards.pop())
#以顺时针方向显示 4 手牌
print("东：",end="")
for i in range(0,13,1):
    print(pack1[i],end="  ")
print("\n南：",end="")
for i in range(0,13,1):
    print(pack2[i],end="  ")
print("\n西：",end="")
for i in range(0,13,1):
    print(pack3[i],end="  ")
print("\n北：",end="")
for i in range(0,13,1):
    print(pack4[i],end="  ")运行结果：
```

运行结果如图 6-25 所示。

```
====================== RESTART: D:/Python36/ch4/p1.py ======================
东：C6   CQ   D10   CK   D3   C10   D7   H2   D4   C3   S10   S7   C5
南：C7   DA   S8   HJ   S2   D9   H3   S5   H4   D5   D6   CA   HK
西：HQ   C4   H5   H8   DJ   C8   D8   C9   H7   H10   C2   H9   H6
北：DQ   DK   SK   CJ   S9   D2   H3   SA   S4   HA   SJ   S6   SQ
```

图 6-25　例 6-25 的运行结果

6.4　时间类模块

Python 中与处理时间相关的模块包括 3 个：time 模块、datetime 模块和 calendar 模块。在介绍这些模块应用前，先说明 Python 语言表示时间的 3 种方式：时间戳、格式化时间字符串和 struct_time 元组。

1. 时间戳

时间戳表示的是从 1970 年 1 月 1 日 00：00：00 开始按秒计算的偏移量，这是基于与

UNIX 操作系统所用的 UTC 时间保持一致。所以，若使用运算 type(time. time())，则系统将返回 float 型数据。

注：UTC(Coordinated Universal Time，协调世界时)，又称世界统一时间、世界标准时间等。UTC 是以原子时的秒长为基础的，从而在时刻上尽量保持精准的一种计时系统。目前，我国采用的是 ISO 8601-1988 标准《数据元和交换格式信息交换日期和时间表示法》。

2. 格式化的时间字符串

Python 是将日期与时间作为对象进行处理，并要求显示日期与时间信息时需要进行适当的格式化。

【例 6-26】 设定当前日期时间的格式为：年-月-日 时：分：秒。

在 IDLE 交互环境中，输入如下命令：

```
>>>import datetime                                  #导入 datetime 模块
>>>datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

本例中的首先导入 datetime 模块，其后才能进行函数调用；这里的 datetime. now()函数(与模块同名)将获取当前日期与时间信息，strftime()函数将显示相应的格式化的时间字符串。其中，符号%导引的就是格式描述信息。

运行结果如图 6-26 所示。

```
>>> import datetime
>>> datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
'2017-12-25 11:56:33'
```

图 6-26 例 6-26 的运行结果

3. struct_time 元组

struct_time 元组总共包含 9 个元素，用于描述有关日期、时间、星期等内容，如表 6-3 所示。

表 6-3 struct_time 元组

索引	属性	取值及其范围
0	tm_year(年)	1900～2099
1	tm_mon(月)	1～12
2	tm_mday(日)	1～31
3	tm_hour(时)	0～23
4	tm_min(分)	0～59
5	tm_sec(秒)	0～59
6	tm_wday(weekday)	0～6(0 表示星期日)
7	tm_yday(一年中的第几天)	1～366
8	tm_isdst(是否为夏时制)	隐含为－1

返回 struct_time 元组的函数主要有 gmtime()、localtime()、strptime()等，下面分别进行说明。

6.4.1 time 模块

time 模块用于处理时间，下面介绍常用的函数及其调用。不过，在调用函数前，必须导入 time 模块，其引用格式如下：

```
>>>import time
```

1. localtime(＜secs＞)

功能：将一个时间戳转换为当前时区的 struct_time 元组，若没有设置参数＜secs＞时则以当前时间为准。

在 IDLE 交互环境中，输入如下命令：

```
>>>time.localtime()
```

运行结果如图 6-27 所示。

```
>>> import time
>>> time.localtime()
time.struct_time(tm_year=2018, tm_mon=1, tm_mday=14, tm_hour=
16, tm_min=56, tm_sec=14, tm_wday=6, tm_yday=14, tm_isdst=0)
```

图 6-27　localtime()函数的运行结果

2. gmtime(＜secs＞)

功能：与 localtime()类似，gmtime(＜secs＞)是将一个时间戳转换为 UTC 时区(0 时区)的 struct_time 元组。

在 IDLE 交互环境中，输入如下命令：

```
>>>time.gmtime()
```

运行结果如图 6-28 所示。

```
>>> import time
>>> time.gmtime()
time.struct_time(tm_year=2018, tm_mon=1, tm_mday=14, tm_hour=
8, tm_min=58, tm_sec=17, tm_wday=6, tm_yday=14, tm_isdst=0)
```

图 6-28　gmtime()函数的运行结果

3. time()

功能：返回当前时间的时间戳。

在 IDLE 交互环境中，输入如下命令：

```
>>>time.time()
```

运行结果如图 6-29 所示。

```
>>> import time
>>> time.time()
1514174453.9042757
```

图 6-29　time()函数的运行结果

4. mktime(＜t＞)

功能：将指定的 struct_time 元组转化为时间戳。

在 IDLE 交互环境中，输入如下命令：

```
>>>time.mktime(time.localtime())
```

运行结果如图 6-30 所示。

```
>>> import time
>>> time.mktime(time.localtime())
1514174499.0
```

图 6-30　mktime()函数的运行结果

5. sleep(<secs>)

功能：设置线程推迟至指定时间后运行，单位为秒。

在 IDLE 交互环境中，输入如下命令：

```
>>>time.sleep(10)
```

以上程序段将强制用户等待 10s 后才能进行操作，即当前线程推迟 10s 后运行。

6. clock()

功能：需要注意该函数的使用限制，在不同系统上的含义不同。在 UNIX 系统上，它返回的是“进程时间”，它是以秒为单位表示的浮点数(时间戳)，而在 Windows 中，第一次函数调用时，返回的是进程运行的实际时间，第二次之后的函数调用是自第一次调用以后到现在的运行时间。

【例 6-27】　clock()函数示例。

源程序如下：

```
import time
if __name__=="__main__":
    time.sleep(2)
    print("clock1:\t%s"%time.clock())
    time.sleep(2)
    print("clock2:\t%s"%time.clock())
    time.sleep(2)
    print("clock3:\t%s"%time.clock())
```

本例中的第一个 clock()输出的是程序运行时间，第二个 clock()、第三个 clock()输出的都是与第一个 clock()相对应的时间间隔。

运行结果如图 6-31 所示。

```
======================= RESTART: D:/Python36/ch6/p1.py =======================
clock1: 6.220394808458493e-07
clock2: 2.015923588670173
clock3: 4.021608096465883
```

图 6-31　例 6-27 的运行结果

7. asctime(<t>)

功能：把一个表示时间的元组或者 struct_time 表示为如下形式：

```
Mon Dec 25 12:03:17 2017
```

如果没有参数，则函数将 time.localtime()作为参数传入。

【例 6-28】　asctime()函数示例。

在 IDLE 交互环境中，输入如下命令：

```
>>>time.asctime()
```

运行结果如图 6-32 所示。

```
>>> import time
>>> time.asctime()
'Mon Dec 25 12:03:17 2017'
```

图 6-32　例 6-28 的运行结果

8. ctime(＜secs＞)

功能：把一个时间戳(按秒计算的浮点数)转化为 time.asctime()的形式。如果没有参数或者参数为 None，则将会隐含 time.time()为参数，其作用等同于 time.asctime(time.localtime(＜secs＞))。

【例 6-29】 ctime()函数示例。

源程序如下：

```
import time
print(time.ctime())
print(time.ctime(time.time()))
print(time.ctime(1508505289.0))
```

运行结果如图 6-33 所示。

```
====================== RESTART: D:/Python36/ch6/p1.py ======================
Mon Dec 25 12:03:54 2017
Mon Dec 25 12:03:55 2017
Fri Oct 20 21:14:49 2017
```

图 6-33　例 6-29 的运行结果

9. strftime(＜format＞,＜t＞)

功能：把一个代表时间的元组或者 struct_time(由 time.localtime()和 time.gmtime()返回)转化为格式化的时间字符串。如果没有指定参数＜t＞，则传入 time.localtime()。如果元组中任何一个元素越界，系统将会抛出 ValueError 异常；参数＜format＞表示格式化的时间字符串，其中的选项如表 6-4 所示。

表 6-4　参数＜format＞中的选项

格　式	含　义
%a	本地简化星期名称
%A	本地完整星期名称
%b	本地简化月份名称
%B	本地完整月份名称
%c	本地相应的日期和时间表示
%d	一个月中的第几天(01～31)
%H	一天中的第几个小时(24 小时制，00～23)

格　　式	含　　义
%I	第几个小时(12 小时制,01～12)
%j	一年中的第几天(001-366)
%m	月份(01～12)
%M	分钟数(00～59)
%p	本地 am 或者 pm 的相应符号
%S	秒(01～61)
%U	一年中的星期数(00～53),星期天作为开始,首个星期天前的天数放在第 0 周
%w	一个星期中的第几天(0～6,0 是星期天)
%W	和%U 基本相同,不同的是%W 以星期一为一个星期的开始
%x	本地相应日期
%X	本地相应时间
%y	去掉世纪的年份(00～99)
%Y	完整的年份
%Z	时区名字(若不存在则为空字符)
%%	"%"字符

在 IDLE 交互环境中,输入如下命令:

```
>>>time.strftime("%Y-%m-%d %X", time.localtime())
```

运行结果如图 6-34 所示。

```
>>> import time
>>> time.strftime("%Y-%m-%d %X", time.localtime())
'2017-12-25 12:04:40'
```

图 6-34　strftime()函数的运行结果

10. strptime(＜string＞,＜format＞)

功能:把一个格式化时间字符串转化为 struct_time 元组,这是 strftime()函数的逆操作。

在 IDLE 交互环境中,输入如下命令:

```
>>>time.strptime("2017-12-24 16:00:00", "%Y-%m-%d %X")
```

运行结果如图 6-35 所示。

```
>>> import time
>>> time.strptime("2017-12-24 16:00:00","%Y-%m-%d %X")
time.struct_time(tm_year=2017, tm_mon=12, tm_mday=24, tm_hour
=16, tm_min=0, tm_sec=0, tm_wday=6, tm_yday=358, tm_isdst=-1)
```

图 6-35　strptime()函数的运行结果

6.4.2 datetime 模块

datetime 模块用于处理日期和时间，下面介绍常用的函数及其调用。不过，在调用函数前，必须导入 datetime 模块，其引用格式如下：

```
>>>import datetime
```

1. now()

功能：获取当前日期和时间。

在 IDLE 交互环境中，输入如下命令：

```
>>>import datetime
>>>datetime.datetime.now()
```

运行结果如图 6-36 所示。

```
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2017, 12, 25, 12, 8, 7, 165135)
```

图 6-36 now()函数的运行结果

2. today()

功能：获取当前日期。

在 IDLE 交互环境中，输入如下命令：

```
>>>datetime.date.today()
```

运行结果如图 6-37 所示。

```
>>> import datetime
>>> datetime.date.today()
datetime.date(2017, 12, 25)
```

图 6-37 today()函数的运行结果

3. timedelta()

功能：获取经过计算的日期

在 IDLE 交互环境中，输入如下命令：

```
>>>datetime.datetime.now()
>>>datetime.datetime.now()-datetime.timedelta(days=2)
```

运行结果如图 6-38 所示。

```
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2017, 12, 25, 12, 9, 17, 443998)
>>> datetime.datetime.now()-datetime.timedelta(days=2)
datetime.datetime(2017, 12, 23, 12, 9, 31, 198837)
```

图 6-38 timedelta()函数的运行结果

4. combine()

功能：获取当天的开始和结束时间。

在 IDLE 交互环境中，输入如下命令：

```
>>>datetime.datetime.combine(datetime.date.today(), datetime.time.min)
>>>datetime.datetime.combine(datetime.date.today(), datetime.time.max)
```

命令中的 datetime. time. min 表示当天开始的时间，datetime. time. max 当天结束的时间。运行结果如图 6-39 所示。

```
>>> import datetime
>>> datetime.datetime.combine(datetime.date.today(), datetime.time.min)
datetime.datetime(2017, 12, 25, 0, 0)
>>> datetime.datetime.combine(datetime.date.today(), datetime.time.max)
datetime.datetime(2017, 12, 25, 23, 59, 59, 999999)
```

图 6-39　combine()

【例 6-30】 获取时间差。

在 IDLE 交互环境中，输入如下命令：

```
>>> (datetime.datetime(2018, 1, 1, 0, 0, 0)-datetime.datetime.now()).total_seconds()
```

运行结果如图 6-40 所示。

```
>>> import datetime
>>> (datetime.datetime(2018,1,1,0,0,0)-datetime.datetime.now()).total_seconds()
556408.198326
```

图 6-40　例 6-30 的运行结果

6.4.3　calendar 模块

calendar 模块中的函数都是处理日历数据的，例如显示字符形式的月历。其中，星期一将隐含为每周的第一天，星期天则是隐含的最后一天。下面逐条说明 calendar 模块中的函数。不过，在调用函数前，必须导入 calendar 模块，其引用格式如下：

```
>>>import calendar
```

1. calendar(<year>,<*w*>,<*l*>,<*c*>)

功能：返回一个多行字符串格式的年历，3 个月为一行，年份由参数<year>指定，每日宽度间隔由参数<*w*>指定，每星期行数由参数<*l*>指定，间隔距离由参数<*c*>指定，每行长度为 $21w+18+2c$。

2. firstweekday()

功能：返回当前每周起始日期的设置。隐含情况是载入 caendar 模块时返回 0，即星期一。

3. isleap(<year>)

闰年判断。若是返回 True，否则返回 false。

4. leapdays(<y_1>,<y_2>)

功能：返回在 y_1～y_2之间的闰年总数。

5. month(<year>,month,<*w*>,<*l*>)

功能：返回一个多行字符串格式的月历，两行标题，一周一行。年份由<year>指定，月份由 month 指定，每日宽度间隔由<*w*>指定，每星期行数由<*l*>指定，每行的长度为 $7w+6$。

6. monthcalendar(<year>,month)

功能：返回一个整数的单层嵌套列表，每个子列表装载代表一个星期的整数。年份

＜year＞和月份 month 外的日期都设为 0，范围内的日子都由该月第几日表示，从 1 开始。

7. monthrange(＜year＞,＜month＞)

功能：返回两个整数。第一个整数是该月的星期几的日期数，第二个整数是该月的日期数。日从 0(星期一)到 6(星期日)记数，月从 1～12 记数。

8. prcal(＜year＞,＜*w*＞,＜*l*＞,＜*c*＞)

功能：与 calendar. calendar(＜year＞,＜*w*＞,＜*l*＞,＜*c*＞)函数相当。

9. prmonth(＜year＞,＜month＞,＜*w*＞,＜*l*＞)

功能：与 calendar. calendar(＜year＞,＜*w*＞,＜*l*＞,＜*c*＞)函数相当。

10. setfirstweekday(weekday)

功能：设置每周的起始日期数，星期表示是 0(星期一)到 6(星期日)。

11. timegm(tupletime)

功能：gmtime()函数的逆操作，接受一个时间元组形式，返回该时刻的时间辍(1970 年元旦后经过的秒数)。

12. weekday(＜year＞,＜month＞,＜day＞)

功能：返回给定日期的日期数。其中，星期表示是 0(星期一)到 6(星期日)，月份表示是 1(一月)到 12(十二月)。

【例 6-31】 调用 celendar()函数生成年历。

源程序如下：

```
import calendar
#设定年份为 2018
cal=calendar.calendar(2018)
#获取年历数据并显示
cal=calendar.HTMLCalendar(calendar.MONDAY)
print(cal.formatyear(2018))
```

运行结果如图 6-41 所示。

```
======================= RESTART: D:\Python36\ch6\p1.py =======================
<table border="0" cellpadding="0" cellspacing="0" class="year">
<tr><th colspan="3" class="year">2018</th></tr><tr><td><table border="0" cellpa
dding="0" cellspacing="0" class="month">
<tr><th colspan="7" class="month">January</th></tr>
<tr><th class="mon">Mon</th><th class="tue">Tue</th><th class="wed">Wed</th><th
class="thu">Thu</th><th class="fri">Fri</th><th class="sat">Sat</th><th class="
sun">Sun</th></tr>
<tr><td class="mon">1</td><td class="tue">2</td><td class="wed">3</td><td class
="thu">4</td><td class="fri">5</td><td class="sat">6</td><td class="sun">7</td>
</tr>
<tr><td class="mon">8</td><td class="tue">9</td><td class="wed">10</td><td clas
s="thu">11</td><td class="fri">12</td><td class="sat">13</td><td class="sun">14
</td></tr>
<tr><td class="mon">15</td><td class="tue">16</td><td class="wed">17</td><td cl
ass="thu">18</td><td class="fri">19</td><td class="sat">20</td><td class="sun">
21</td></tr>
<tr><td class="mon">22</td><td class="tue">23</td><td class="wed">24</td><td cl
ass="thu">25</td><td class="fri">26</td><td class="sat">27</td><td class="sun">
28</td></tr>
<tr><td class="mon">29</td><td class="tue">30</td><td class="wed">31</td><td cl
ass="noday"> </td><td class="noday"> </td><td class="noday"> </t
d><td class="noday"> </td></tr>
</table>
</td><td><table border="0" cellpadding="0" cellspacing="0" class="month">
<tr><th colspan="7" class="month">February</th></tr>
<tr><th class="mon">Mon</th><th class="tue">Tue</th><th class="wed">Wed</th><th
class="thu">Thu</th><th class="fri">Fri</th><th class="sat">Sat</th><th class="
sun">Sun</th></tr>
<tr><td class="noday"> </td><td class="noday"> </td><td class="noday"
```

图 6-41　显示年历的字符串格式(截图未完)

在图 6-41 中呈现的内容并不是真正的年历，为此还要进行如下 3 步操作：

(1) 使用 Windows 的复制技术将全部显示内容放入剪贴板中。

(2) 使用 Windows 的记事本将剪贴板内容保存为 HTML 文档，只要指定文件扩展名为.HTML 即可。

(3) 在 IE 浏览器中打开 HTML 文档就可以得到年历效果，如图 6-42 所示。

```
                                  2018
        January                    February                    March
MonTueWedThuFriSatSunMonTueWedThuFriSatSunMonTueWedThuFriSatSun
1  2  3  4  5  6  7           1  2  3  4           1  2  3  4
8  9  10 11 12 13 14 5  6  7  8  9  10 11 5  6  7  8  9  10 11
15 16 17 18 19 20 21 12 13 14 15 16 17 18 12 13 14 15 16 17 18
22 23 24 25 26 27 28 19 20 21 22 23 24 25 19 20 21 22 23 24 25
29 30 31             26 27 28             26 27 28 29 30 31
        April
MonTueWedThuFriSatSun         May                    June
                  1  MonTueWedThuFriSatSunMonTueWedThuFriSatSun
2  3  4  5  6  7  8     1  2  3  4  5  6              1  2  3
9  10 11 12 13 14 15 7  8  9  10 11 12 13 4  5  6  7  8  9  10
16 17 18 19 20 21 22 14 15 16 17 18 19 20 11 12 13 14 15 16 17
23 24 25 26 27 28 29 21 22 23 24 25 26 27 18 19 20 21 22 23 24
30                   28 29 30 31          25 26 27 28 29 30
        July
MonTueWedThuFriSatSun        August                September
                  1  MonTueWedThuFriSatSunMonTueWedThuFriSatSun
2  3  4  5  6  7  8        1  2  3  4  5                 1  2
9  10 11 12 13 14 15 6  7  8  9  10 11 12 3  4  5  6  7  8  9
16 17 18 19 20 21 22 13 14 15 16 17 18 19 10 11 12 13 14 15 16
23 24 25 26 27 28 29 20 21 22 23 24 25 26 17 18 19 20 21 22 23
30 31                27 28 29 30 31       24 25 26 27 28 29 30
                                                  December
        October              November        MonTueWedThuFriSatSun
MonTueWedThuFriSatSunMonTueWedThuFriSatSun                   1  2
1  2  3  4  5  6  7           1  2  3  4  3  4  5  6  7  8  9
8  9  10 11 12 13 14 5  6  7  8  9  10 11 10 11 12 13 14 15 16
15 16 17 18 19 20 21 12 13 14 15 16 17 18 17 18 19 20 21 22 23
22 23 24 25 26 27 28 19 20 21 22 23 24 25 24 25 26 27 28 29 30
29 30 31             26 27 28 29 30       31
```

图 6-42 显示年历

6.5 os 模块

在使用 Python 编程时，通常要对文件或文件夹进行访问，这时就必须利用 os 模块。os 模块模拟操作系统的文件管理，主要目标是实现文件的“按名存取”。

6.5.1 常用函数

os 模块包含许多与操作系统相关的操作功能，常用函数如表 6-5 所示。

表 6-5 常用函数

函　　数	说　　明
name	判断当前系统的操作平台
getcwd()	得到当前文件夹
listdir()	取消指定文件夹中所有的文件名和文件夹名

续表

函　　数	说　　明
remove()	删除指定文件
rmdir()	删除指定文件夹
mkdir()	创建文件夹
path.isfile()	判断指定对象是否为文件，若是返回 True，否则 False
path.isdir()	判断指定对象是否为文件夹，若是返回 True，否则 False
path.exists()	检验指定的对象是否存在，若是返回 True，否则 False
path.split()	返回路径的文件夹和文件名
system()	执行 shell 命令
chdir()	改变文件夹到指定文件夹
path.getsize()	获得文件的大小，若为文件夹则返回 0
path.abspath()	获得绝对路径
path.join(path, name)	连接文件夹和文件名
path.basename(path)	返回文件名
path.dirname(path)	返回文件路径

【例 6-32】 显示指定文件夹中的所有文件名和文件夹名。

在 IDLE 交互环境中，输入如下命令：

```
>>>import os
>>>os.listdir("D:\Python36")
```

运行结果如图 6-43 所示。

```
>>> import os
>>> os.listdir("D:\Python36")
['ch1', 'ch2', 'ch3', 'ch4', 'ch5', 'ch6', 'ch7', 'ch8', 'ch9
', 'cha', 'chb', 'chc', 'DLLs', 'Doc', 'include', 'Lib', 'lib
s', 'LICENSE.txt', 'NEWS.txt', 'python.exe', 'python.pdb', 'p
ython3.dll', 'python36.dll', 'python36.pdb', 'python36_d.dll'
, 'python36_d.pdb', 'python3_d.dll', 'pythonw.exe', 'pythonw.
pdb', 'pythonw_d.exe', 'pythonw_d.pdb', 'python_d.exe', 'pyth
on_d.pdb', 'Scripts', 'tcl', 'Tools', 'vcruntime140.dll']
```

图 6-43　例 6-32 的运行结果

【例 6-33】 创建文件夹并检测。

在 IDLE 交互环境中，输入如下命令：

```
>>>os.mkdir("tmp")
>>>os.path.isdir("tmp")
```

运行结果如图 6-44 所示。

```
>>> import os
>>> os.mkdir("tmp")
>>> os.path.isdir("tmp")
True
```

图 6-44　例 6-33 的运行结果

本例中的第 1 行在当前文件夹位置创建一个名为 tmp 的文件夹，第 2 行检测该文件夹是否创建成功，结果显示为 True。

【例 6-34】 将指定路径分解为文件夹和文件名两部分，且用元组表示。

在 IDLE 交互环境中，输入如下命令：

```
>>>os.path.split("D:\\Python36\\ch6\\p1.py")
```

运行结果如图 6-45 所示。

```
>>> import os
>>> os.path.split("D:\\Python36\\ch6\\p1.py")
('D:\\Python36\\ch6', 'p1.py')
```

图 6-45　例 6-34 的运行结果

从运行结果中可以发现，path.split()将路径分解为文件夹 D:\Python36 和文件名 p1.py 两部分，并用用元组表示。要注意的是，字符串中的\\符号是转义字符，即\符号本身。

【例 6-35】 查看文件的时间信息。

在 IDLE 交互环境中，输入如下命令：

```
>>>os.path.getmtime("F: \\Python36\\P1.py")        #输出文件的创建时间
>>>os.path.getatime("F: \\Python36\\P1.py")        #输出文件的修改时间
>>>os.path.getctime("F: \\Python36\\P1.py")        #输出文件的保存时间
```

运行结果如图 6-46 所示。

```
>>> import os
>>> os.path.getmtime("D:\\Python36\\ch6\\p1.py")
1514179695.9175196
>>> os.path.getatime("D:\\Python36\\ch6\\p1.py")
1514172934.585154
>>> os.path.getctime("D:\\Python36\\ch6\\p1.py")
1514172934.585154
```

图 6-46　例 6-35 的运行结果

【例 6-36】 查看文件或文件夹的大小。

在 IDLE 交互环境中，输入如下命令：

```
>>>os.path.getsize("D:\\python36\\python.pdb")
>>>os.path.getsize("D:\\python36\\python36.pdb")
```

本例中的存储空间是以字节为单位的，所以 9097216B 就是 8884(9097216/1024)KB，后者才是 Windows 文件管理器中的存储显示格式。

运行结果如图 6-47 所示。

```
>>> import os
>>> os.path.getsize("D:\\python36\\python.pdb")
446464
>>> os.path.getsize("D:\\python36\\python36.pdb")
9097216
```

图 6-47　例 6-36 的运行结果

6.5.2　程序示例

以下程序是通过调用函数实现的，所以只要安排正确的调用时序即可。

【例 6-37】 查找文件夹中最新的文件和文件夹。

源程序如下：

```
#导入 os 模块
import os
#定义函数 new_file()
def new_file(test_folder):
    filenames=os.listdir(test_folder)
    filenames.sort(key=lambda fn:os.path.getmtime(test_folder+"\\"+fn))
    file_path=os.path.join(test_folder,filenames[-1])
    return file_path
#主函数及其函数调用
print("最新文件:")
print(new_file("D:\\Python36"))
```

本例中的第 4～8 行定义 new_file()函数，第 5 行获取指定文件夹中的全部文件名(以列表 filenames 表示)，第 6 行使用 lambda 表达式将文件名按从小到大排序，第 7 行获取列表 filenames 中的最后一个元素(即最新文件)，第 8 行将其返回。主函数在显示提示信息后，调用 new_file()函数并找出最新文件。

运行结果如图 6-48 所示。

```
======================= RESTART: D:\Python36\ch6\p1.py =======================
最新文件:
D:\Python36\ch6
```

图 6-48 例 6-37 的运行结果

注意：运行结果中 ch6 是文件夹名称，本例程序是存放在该文件夹中的。

【例 6-38】 查找文件夹中最新的 5 个文件和文件夹。

源程序如下：

```
import os
test_folder="D:\\Python36"              # 指定文件夹
filenames=os.listdir(test_folder)
filenames.sort(key=lambda fn:os.path.getmtime(test_folder+"\\"+fn))
for i in range(-1,-6,-1):
    file_path=os.path.join(test_folder,filenames[i])
    print(file_path)
```

本例中使用循环结构来显示最新的 5 个文件，其中包括扩展名为.tmp 的临时文件，通常临时文件就是刚建立的新文件。

运行结果如图 6-49 所示。

```
======================= RESTART: D:\Python36\ch6\p1.py =======================
D:\Python36\ch6
D:\Python36\ch5
D:\Python36\ch4
D:\Python36\ch3
D:\Python36\ch1
```

图 6-49 例 6-38 的运行结果

注意：运行结果中显示的均是文件夹名称。

【例 6-39】 获得当前程序的文件名和文件夹信息。

源程序如下：

```
import os
#导入 sys 模块
import sys
if __name__=="__main__":
    print(os.path.realpath(sys.argv[0]))
    print(os.path.split(os.path.realpath(sys.argv[0])))
    print(os.path.split(os.path.realpath(sys.argv[0]))[0])
```

运行结果如图 6-50 所示。

```
====================== RESTART: D:\Python36\ch6\p1.py ======================
D:\Python36\ch6\p1.py
('D:\\Python36\\ch6', 'p1.py')
D:\Python36\ch6
```

图 6-50 例 6-39 的运行结果

6.6 sys 模 块

sys 模块模拟操作系统的系统管理，内容包括操作系统平台及其版本的信息、模块搜索路径、模块表、异常信息、命令行参数、标准输入输出流、程序退出等。

6.6.1 常用函数

sys 模块的常用函数如表 6-6 所示。

表 6-6 常用函数

函 数	说 明
argv	从程序外部向程序传递参数
exit([<arg>])	退出程序运行，arg 为 0 时表示正常退出
getdefaultencoding()	获取系统当前编码，隐含为 ASCII
setdefaultencoding()	设置隐含编码，执行 dir(sys)后将不会有该函数。若执行 reload(sys)后，再执行 setdefaultencoding("utf_8")则将隐含设置为 UTF-8
getfilesystemencoding()	获取文件系统使用的编码方式
path	获取指定模块搜索路径的字符串集合，若将模块放在指定个路径中，则可以在 import 语句执行时找到
platform	获取当前操作系统平台
stdin	标准输入流
stdout	标准输出流
stderr	标准错误流

【例 6-40】 查看模块搜索路径。

若要显示机器中安装 Python 的路径情况，则可以在 IDLE 交互环境，输入如下命令：

```
>>>import sys
>>>sys.path
```

运行结果如图 6-51 所示。

```
>>> import sys
>>> sys.path
['D:\\Python36\\ch4', 'D:\\Python36\\Lib\\idlelib', 'D:\\Pyth
on36\\python36.zip', 'D:\\Python36\\DLLs', 'D:\\Python36\\lib
', 'D:\\Python36', 'D:\\Python36\\lib\\site-packages']
```

图 6-51　例 6-40 的运行结果

【例 6-41】 设置模块搜索路径。

在 IDLE 交互环境中，输入如下命令：

```
>>>sys.path.append("D:\\Pathon36\\PathonPDF")
>>>sys.path
```

运行结果如图 6-52 所示。

```
>>> import sys
>>> sys.path.append("D:\\Pathon36\\PathonPDF")
>>> sys.path
['', 'D:\\Python36\\Lib\\idlelib', 'D:\\Python36\\python36.zip',
'D:\\Python36\\DLLs', 'D:\\Python36\\lib', 'D:\\Python36', 'D:\\
Python36\\lib\\site-packages', 'D:\\Pathon36\\PathonPDF']
```

图 6-52　例 6-41 的运行结果

6.6.2　命令行参数

利用 sys 模块中的命令行参数 argv 可以获取用户在命令行窗口或 IDLE shell 环境运行程序时给出的参数列表。其中，argv 是一个列表，列表的第一个元素 argv[0]是程序文件名，从 argv[1]开始才是程序文件后面跟随的参数部分。

Python 中也可以所用 sys.argv 来获取命令行参数：

(1) sys.argv 是命令行参数列表。

(2) len(sys.argv)是命令行参数个数。

说明：

(1) sys.argv[0]表示程序名。

(2) sys.argv[1]、sys.argv[2]和 sys.argv[3]用于提供程序所需的 3 个数据。

进入命令提示符窗口的操作如下：右击“开始”菜单，从弹出的快捷菜单中选中“运行”选项，在弹出的“运行”对话框中，输入命令 cmd，其后系统将显示进入命令指示符窗口，然后进行操作。

【例 6-42】 命令提示符窗口使用示例。

源程序如下：

```
import sys
a=int(sys.argv[1])            # 将参数 sys.argv[1]的值赋给变量 a
b=int(sys.argv[2])
```

```
c=int(sys.argv[3])
print(a,b,c,len(sys.argv))
```

将上述程序命名为 p1.py 后,可在命令提示符窗口中输入如下命令:

```
Python p1.py 1 2 3<Enter>
```

运行结果如图 6-53 所示。

```
C:\Windows\system32\cmd.exe
D:\Python36>python p1.py 1 2 3
1 2 3 4

D:\Python36>
```

图 6-53　例 6-42 的运行结果

在运行结果中发现,显示的 1,2,3 就是输入的 3 个数据,显示的 4 表示参数个数。

【例 6-43】　命令提示符窗口使用示例。

源程序如下:

```
import sys
r=int(sys.argv[1])
cir=2 * 3.14 * r
area=3.14 * r * r
v=4/3 * r * 3.14 * r * r * r
print("圆面积: ",cir)
print("圆周长: ",area)
print("球体积: ",v)
```

输入数据为 2,是通过附在命令后实现的。运行结果如图 6-54 所示。

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>d:

D:\>cd\Python36\ch6

D:\Python36\ch6>Python p1.py 2
圆面积:  12.56
圆周长:  12.56
球体积:  66.98666666666666

D:\Python36\ch6>
```

图 6-54　例 6-43 的运行结果

6.7　自定义模块

由于 Python 也具有内存自动回收机制,从而在退出 Python 解释器后,原来的数据定义和程序环境将会全部丢失。另一方面,对一个大型程序而言,必须进行功能划分,不可能

由一个程序文件来完成所有功能。为了满足这些需要,Python 提供自定义模块的方法,允许在一个文件(即模块)中进行函数定义和常量声明,然后导入模块,最后就可以从模块中调用函数和常量。

6.7.1 主模块

1. 变量__name__

Python 中的模块就是任何程序文件,文件名就是模块名加上.py 作为文件扩展名。模块名(作为一个字符串)可以由全局变量__name__来表示。

【例 6-44】 变量__name__示例。

(1) 在当前文件夹中建立名为 p1.py 的文件。

```
#自定义模块
r=int(input("r="))
print("园面积:",3.14*r*r)
print("圆周长:",2*3.14*r)
print("球体积:",4*3.14*r*r*r/3)
```

(2) 然后在 IDLE 交互环境中输入如下命令:

```
>>>import p1
>>>p1.__name__
```

运行结果如图 6-55 所示。

```
>>> import p1
r=2
园面积:  12.56
圆周长:  12.56
球体积:  33.49333333333333
>>> p1.__name__
'p1'
```

图 6-55 例 6-44 的运行结果

从运行结果中可以发现,前面的一个程序文件可以作为模块,并通过变量__name__来表示该文件。当然,这里产生的显示结果显然与模块构造理念是不一致的。

注意:导入自定义模块时不能有文件扩展名.py。

2. 主模块及其测试代码

变量__name__经常用于代码测试。在程序测试阶段,可以将一些测试代码放在对应变量__name__的 if 测试代码。测试时将该文件作为主程序直接运行,这样测试代码就可以执行。测试完成后,该文件作为模块在其他程序中被导入使用时,测试代码就不会被执行了。

任何文件都可以作为模块库的一个模块被其他模块导入,模块导入后才被执行。如果要把某个模块文件作为主程序,可在主程序代码的最后添加一个 if 语句,这是该模块作为主程序的标志。这个 if 语句的一般引用格式如下:

```
if __name__=="__main__":
    <语句块>
```

【例 6-45】 主模块及其代码测试示例。

(1) 列出例 6-11 中的源程序。

```
from decimal import *
getcontext().prec=32
print("精度 32 位: \n\t",Decimal(2)/Decimal(7))
```

(2) 添加测试代码。

含测试代码的源程序如下:

```
from decimal import *
getcontext().prec=32
print("精度 32 位: \n\t",Decimal(2)/Decimal(7))
#测试代码
if __name__=="__main__":
    print("精度 32 位: \n\t",Decimal(2)/Decimal(7))
```

本例中的第 3、6 行将显示相同的结果,但是第 6 行属于测试代码。

运行结果如图 6-56 所示。

```
==================== RESTART: D:/Python36/ch6/ch6_11.py ====================
精度32位:
        0.28571428571428571428571428571429
精度32位:
        0.28571428571428571428571428571429
```

图 6-56

6.7.2 自定义模块示例

下面给出自定义模块的要求,内含一个常量和 3 个函数:

(1) 常量 str="这是自定义模块";

(2) 含 1 个参数的函数 square(a)计算正方形面积;

(3) 含 2 个参数的函数 squareness(a,b)计算矩形面积;

(4) 含 3 个参数的函数 cube(a,b,c)计算立方体体积。

【例 6-46】 自定义模块示例。

(1) 将自定义模块命名为 ch6_46.py。

源程序如下:

```
#自定义模块示例
str="这是自定义模块"
def square(a):
    return a*a
def squareness(a,b):
    return a*b
def cube(a,b,c):
    return a*b*c
#测试代码
if __name__=="__main__":
    a=10
    b=20
    c=30
```

```
    print("常量: \t\t",str)
    print("正方形面积: \t",square(a))
    print("矩形面积: \t",squareness(a,b))
    print("立方体体积: \t",cube(a,b,c))
```

(2) 将主程序命名为 p1.py

源程序如下：

```
#导入自定义模块 ch6_46
import ch6_46
#输入数据
a=int(input("a="))
b=int(input("b="))
c=int(input("c="))
#调用自定义模块 ch6_46 的一个常量和 3 个函数
print("常量: \t\t",ch6_46.str)
print("正方形面积: \t",ch6_46.square(a))
print("矩形面积: \t",ch6_46.squareness(a,b))
print("立方体体积: \t",ch6_46.cube(a,b,c))
```

运行结果如图 6-57 所示。

```
======================= RESTART: D:/Python36/ch6/p1.py =======================
a=10
b=20
c=30
常量:                这是自定义模块
正方形面积:          100
矩形面积:            200
立方体体积:          6000
```

图 6-57　例 6-46 的运行结果

6.7.3　Python 编译文件

为减少重复翻译模块的次数，以便提高程序调用模块的速度。Python 将在__pycache__文件夹中存放每个被翻译模块的字节码文件，其扩展名是.pyc。例如，在 Python 3.6 版本中，文件 ch6_46.py 编译后的文件将命名为__pycache__/ch6_46.cpython-36.pyc，其中 cpython-36 就是用于识别的软件版本信息。经过翻译的模块文件是跨平台无关的，即不同系统平台都可以调用。

6.8　自 定 义 包

如果说模块是实现软件功能划分的手段，那么包则是将若干相关模块重组成一个较大型的程序，其中包括程序文件的组织结构。下面通过一个实例进行说明。

6.8.1　包与模块的组织结构

Python 允许将多个模块组成为一个包，其文件组织形式是将包名与文件夹对应，将模块文件全部存放到该包名文件夹中。在调用模块前，可以使用 import 语句导入包，也可以

使用 from…import 语句导入包中的指定模块。当然,若包中还有子包,则可在包名文件夹下建立子包名文件夹,其中可存放模块文件。

下面就是例 6-47 中自定义包的文件组织,如图 6-58 所示。



图 6-58　包的文件组织

从图 6-58 中可以发现,在包名文件夹 ch6 下存放了 3 个模块文件 module_1. py、module_2. py、module_3. py 和存放已编译文件的__pycache__文件夹。注意,主程序必须存放在 ch6 文件夹的同一层中,即 D:\Python36 文件夹。

6.8.2　包与模块的导入

使用 import 语句可以导入包中的模块,下面说明常用的几种导入方法。

(1) 若导入包中的全部模块,可使用如下格式:

```
import <包名> from *
```

在使用这种格式前,需要在包的__init__. py 文件中定义一个名为__all__的列表变量,其中包含的全部模块名,这样才能导入包中的全部模块。若没有定义__all__变量,则这条语句将导致 AttributeError 异常。

【例 6-47】　AttributeError 异常示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>from ch6 import *
>>>ch6.module_1.const_1
```

本例中,由于没有__init__. py 文件,即使有该文件也必须保证完全正确,所以导入包中的全部模块不能成功。

运行结果如图 6-59 所示。

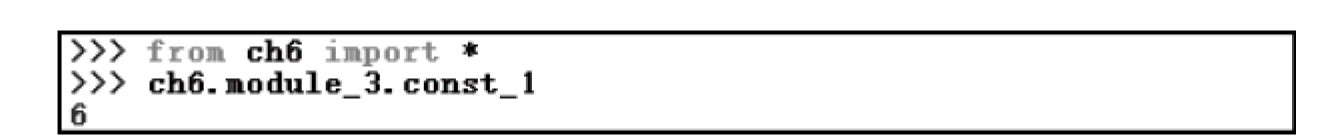


图 6-59　例 6-47 的运行结果

运行结果中的数据 6 取自于模块 module_3. py 中定义的第一个常量 const_1。由于该模块存放在 ch6 文件夹中,所以 ch6 将是包名。请见后面的源程序。

(2) 若导入包中的指定模块,可使用如下格式:

```
import <包名.模块 1>, …, <包名.模块 n>
```

(3) 若导入包中的一个模块,可使用如下格式:

```
import <包名.模块>
```

在导入时要注意两点，一是<模块名>中不能有后缀.py，二是<模块名>可以使用绝对路径。

6.8.3 自定义包示例

创建包文件夹 ch6 作为包名，在其中存放 3 个模块文件，其中前两个是用于函数调用的模块，第 3 个是用于常量调用的模块。另外，主程序必须存放在包文件夹 ch6 的同一层中。

【例 6-48】 自定义包示例。

（1）文件 module_1.py 作为第一个模块，其中定义了 num_prime(<n>)函数。num_prime(<n>)函数的作用是找出 2～n 的全部素数，并以列表形式返回。

```
#定义函数 num_prime()
def num_prime(n):
    p_list=[]
    for m in range(2,n+1,1):
        k=2
        flag=0
        while k<m and flag==0:
            if m%k==0 : flag=1
            k=k+1
        if flag==0 : p_list.append(m)
    return p_list
#测试代码
if __name__=="__main__":
    p=num_prime(10)
    print("显示素数列表: \n",p)
```

本例中的测试代码可以在当前程序中直接调用函数并进行调试，避免在不同程序文件之间来回切换。Python 提供这种程序调试方法，能够提高编程效率。由于模块可以任意导入和调用，所以通常建议使用 if __name__=="__main__"语句，以便保证模块在被 import 语句导入前是能够独立运行的。当然，测试代码对任何形式的函数调用是没有影响的。

运行结果如图 6-60 所示。

```
==================== RESTART: D:/Python36/ch6/module_1.py ====================
显示素数列表:
 [2, 3, 5, 7]
```

图 6-60 例 6-48 的运行结果 1

（2）文件 module_2.py 作为第二个模块，其中定义 num_complete(<n>)函数。num_complete(<n>)函数的作用是找出 1～n 的全部完备数，并以列表形式返回。

```
#定义函数 num_complete()
def num_complete(n):
    p_list=[]
    for m in range(2,n+1,1):
```

```
        mf=1
        for k in range(2,m):
            if m%k==0 : mf=mf+k
        if m==mf : p_list.append(m)
    return p_list
#测试代码
if __name__=="__main__":
    p=num_complete(1000)
    print("显示完备数列表：\n",p)
```

运行结果如图 6-61 所示。

```
==================== RESTART: D:/Python36/ch6/module_2.py ====================
显示完备数列表：
 [6, 28, 496]
```

图 6-61　例 6-48 的运行结果 2

(3) 文件 module_3.py 作为第 3 个模块，其中定义 3 个常量。

```
#定义 3 个常量
const_1=6
const_2=28
const_3=496
#测试代码
if __name__=="__main__":
    print("第 1 个常量：", const_1)
    print("第 2 个常量：", const_2)
    print("第 3 个常量：", const_3)
```

运行结果如图 6-62 所示。

```
==================== RESTART: D:/Python36/ch6/module_3.py ====================
第1个常量： 6
第2个常量： 28
第3个常量： 496
```

图 6-62　例 6-48 的运行结果 3

众所周知，程序错误若传播则负面影响非常大，所以在编程过程中要尽量保证程序正确。虽然这个程序非常简单，但通过测试代码来保证程序正确也是必要的。

创建包文件夹 example 并调试好 3 个模块后，就可编写主程序进行引用了。

(4) 主程序顺序导入包中的 3 个模块，并分别调用模块中的两个函数和 3 个常量。

```
#导入包中的第一个模块
import ch6.module_1
my_list=[]
#调用模块函数 num_prime()
my_list=ch6.module_1.num_prime(10)
length=len(my_list)
print("素数：")
for i in range(0,length,1):
```

```
    print(my_list[i],end="\t")
print("\n 完备数: ")

#导入包中的第二个模块
import ch6.module_2
my_list=[]
#调用模块函数 num_complete()
my_list=ch6.module_2.num_complete(1000)
length=len(my_list)
for i in range(0,length,1):
    print(my_list[i],end="\t")
print("\n 完备数等式: ")

#导入包中的第 3 个模块
import ch6.module_3
#调用 3 个常量 const_1 、const_2 和 const_3
print(ch6.module_3.const_1,"=1+2+3")
print(ch6.module_3.const_2,"=1+2+4+7+14")
print(ch6.module_3.const_3,"=1+2+4+8+16+31+62+124+248")
```

运行结果如图 6-63 所示。

```
======================== RESTART: D:/Python36/p1.py ========================
素数:
2       3       5       7
完备数:
6       28      496
完备数等式:
6 =1+2+3
28 =1+2+4+7+14
496 =1+2+4+8+16+31+62+124+248
```

图 6-63　例 6-48 的运行结果 4

为简化书写,可以一次导入全部模块,例如:

```
import ch6.module_1, ch6.module_2, ch6.module_3
```

习　题　6

一、简答题

1. 简述模块程序设计思想。
2. 如何导入模块? 如何引用模块中的函数和常量。
3. Python 模块的搜索路径包含哪些内容?
4. 简述 math 模块和 cmath 模块的主要函数和常量。
5. 简述 decimal 模块的主要功能。
6. 简述 fractions 模块的主要功能。
7. 简述 random 模块的主要功能。
8. 简述 time 模块、datetime 模块和 calendar 模块的主要函数和常量。
9. 简述 os 模块的主要功能。

10. 简述 sys 模块的主要功能。

11. 如何自定义模块？如何引用自定义模块中的函数和常量？

12. 如何自定义包？如何建立包与模块的文件结构？

13. 如何引用自定义模块中的函数和常量？

二、编程题

1. 调用 math 模块中的 sqrt()和 factorial()函数，分行输出 10～20 的全部平方根和阶乘。

2. 调用 decimal 模块计算 1＋1/2＋1/3＋…＋1/10，要求输出 24 个小数位。

3. 调用 fractions 模块计算 1＋1/2＋1/3＋…＋1/10，要求输出为分数数据。

4. 调用 os 模块显示出指定文件夹中的全部文件。

5. 定义模块计算 100 以内的所有斐波那契数之和，并编写主程序进行模块调用。

6. 设定两个正整数 m 和 n，且 $m<n$，编写 4 个模块：

(1) 模块 1 计算 m～n 的整数和；

(2) 模块 2 计算 m 和 n 的最大公约数；

(3) 模块 3 计算 m 和 n 的最小公倍数；

(4) 模块 4 设定字符串“这是自定义包”作为常量。

编程分别定义包与模块，并合理地组织文件夹结构。

第 7 章　数 据 文 件

Python 语言通过使用数据文件来实现高效的输入输出操作。本章首先介绍数据文件的概念，接着介绍文件的打开与关闭，以及如何读写文本文件和二进制文件，最后介绍能够实现文件读写操作的 struct 模块、fileinput 模块和 codecs 模块。

7.1　文 件 概 述

计算机的文件是由一组相关符号合成的，但这里涉及的是数据文件，即只是为运行程序提供数据的文件。

7.1.1　引言

计算机文件与日常文件的主要区别是载体不同，计算机文件主要使用磁盘作为载体来存储信息，所以计算机文件是具有符号名的，在逻辑上具有完整意义的一组相关信息项的有序序列。其中，数据项是构成文件内容的基本单位。

前面章节中介绍过为程序提供数据的 3 种方式，如表 7-1 所示。

表 7-1　提供数据的 3 种方式

提供数据方式	语　　句	缺　　点
静态初始化	n=10	使程序不能通用
赋值	n=2+8	不易提供大量数据，且增加程序书写量
键盘输入	input(n)	不易提供大量数据，且出错后无法修改

基于以上提供数据的 3 种方式均存在不足，Python 语言提供数据文件以便程序调用，以支持对大量数据的灵活使用。

7.1.2　文件分类

数据文件种类繁多，下面分别介绍标准输入文件与标准输出文件、ASCII 文件与二进制文件、流式文件与记录式文件以及顺序存取文件与随机存取文件。

1. 标准输入文件与标准输出文件

由于 Python 语言没有输入输出语句，只能通过函数调用来实现，可以调用的输入输出函数主要是 input()和 print()。一般而言，标准输入文件是指键盘，由 input()函数所用；标准输出文件是指显示器，由 print()函数所用。

2. ASCII 文件与二进制文件

从文件编码方式来看，文件可分为 ASCII 文件和二进制文件。

ASCII 文件也称为文本文件，这种文件存放一个字符时对应一个字节的存储空间，并用

对应的 ASCII 码进行表示。例如,信息 12345 的存储形式如图 7-1 所示。

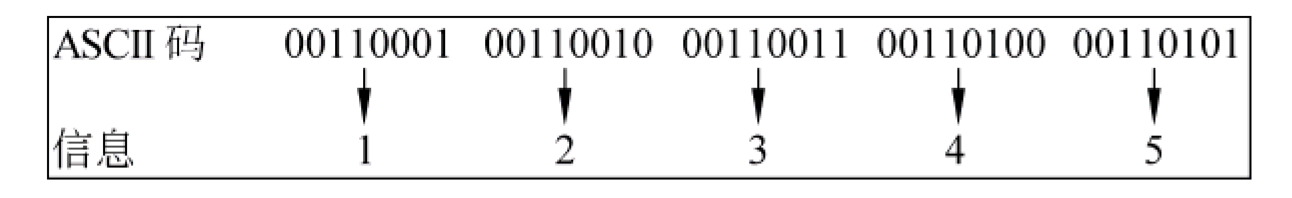


图 7-1 整数 12345 的 ASCII 文件存储

整数 12345 共占用 5B 空间。ASCII 文件可以直接以字符形式显示,例如 Python 源程序文件就是 ASCII 文件,这种按字符显示的方式则可以直接阅读。

二进制文件是按二进制编码方式来存放文件的。例如,整数 12345 的存储形式如图 7-2 所示。

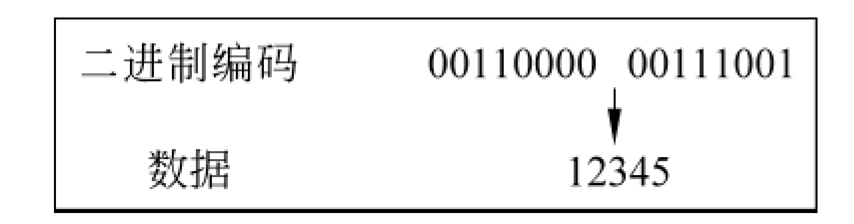


图 7-2 整数 12345 的二进制文件存储

由于整数 12345 转换成二进制数为 11000000111001,高位还要补 00 才占 2B 空间,非常节省存储空间。虽然二进制文件也是可以显示的,但是显示的内容不是字符形式,而是无法阅读的二进制代码。实际上,Python 语言在处理二进制文件时,并没有区分其文件类型(例如 ASCII 文件与二进制文件),都看成是字符流,并按字节为基本单位进行处理,由程序控制字符流的开始和结束,而不受物理符号(例如回车符)的控制,所以二进制文件也是流式文件。

由于 ASCII 文件只实际使用七位编码,所以 ASCII 文件中的每个字节中的最高位都是 0,而二进制文件则是将 1B 长度全部加以利用。

从信息构成来看,ASCII 文件与二进制文件都是由 0 和 1 构成的编码序列。但是由于文件打开方式不同,所以对于 0 和 1 编码的处理也就不同。如果按照 ASCII 模式打开,则在打开时就将进行转换,即将每个字节对应转换成 ASCII 码中的字符;如果按照二进制模式打开,则不会进行任何转换,也没有必要进行转换。

ASCII 文件和二进制文件在编辑时,所使用的方式是不同的。例如,使用 ASCII 文件进行编辑的最小单位是字节,而二进制文件编辑的最小单位是二进制中的位。当然,人们不会以手工操作方式对二进制文件进行编辑。

ASCII 文件和二进制文件的比较如下。

(1) ASCII 文件便于对字符进行逐个处理,也便于输出字符。但一般占存储空间较多,而且要花费转换时间;二进制文件可以节省外存空间和转换时间,但一个字节并不对应一个字符,不能直接输出字符形式。

(2) 一般中间结果数据需要暂时保存在外存上,以后需要调入内存时,则用二进制文件保存。

注意: 在读入一个文本文件时,Python 系统要将 ASCII 码转换成相应的二进制码;而在文件以文本方式写入磁盘时,也要把二进制码转换成相应的 ASCII 码,所以文本文件的读写要花费较多的转换时间,而二进制文件的读写过程不存在数制转换问题。

3. 流式文件与记录式文件

从文件组织方式来看,文件可分为流式文件与记录式文件。

在 Python 语言中,文件是以字符形式、字节存储为基本单位的。在输入输出时,数据流的开始与结束只受程序控制而不受物理符号(例如回车符)控制。例如,在输出时就不会以回车符来作为记录的分隔符。实际上,Python 语言中的文件并不是由记录构成,这种文件被称为流式文件。

记录式文件是由若干逻辑记录组成的,每条逻辑纪录又是由许多数据项组成的。例如,一个学生成绩文件由若干学生记录组成,而每条记录是由学号、姓名、计算机成绩等数据项组成的。

Python 语言中只有流式文件,没有记录式文件。流式文件的基本单位是字符而不是记录,它是有序字符序列的集合,文件长度是该文件所包含的字符个数,所以流式文件也称为字符流文件。

4. 顺序存取文件与随机存取文件

从文件存取方式来看,文件可分为顺序存取文件与随机存取文件。

顺序存取文件是按其在文件中的逻辑顺序依次存取的,只能从前到后顺序访问;随机存取文件是没有顺序存取的限制的,可以进行随机定位并进行访问。

7.2 打开文件与关闭文件

在 Python 中对文件的操作通常按照以下 3 个步骤进行。

(1) 使用 open()函数打开(或建立)文件,返回一个 file 对象。

(2) 使用 file 对象调用读写函数对文件进行读写操作。在这个过程中,从磁盘文件中取出数据的过程称为读取操作,将数据存入磁盘文件的过程称为写入操作。

(3) 使用 file 对象的 close()函数关闭文件。

7.2.1 打开文件

在计算机中,打开文件具有两个含义。一是将指定文件从磁盘装入内存,二是将指定文件与一个文件对象(或指针)建立关联,以方便程序进行调用。open()函数用于打开文件,该函数需要一个字符串形式的(绝对或相对)路径来指定要打开的文件,并返回一个文件对象。其一般调用格式如下:

```
<fileobj>=open(<filename>[, <mode>[, <buffering>=n]])
```

说明:

(1) <fileobj>表示文件对象,用于与指定的数据文件建立关联。

(2) <filename>表示文件名,可以使用相对路径和绝对路径来指定数据文件。

(3) <mode>表示文件操作和类型,文件操作有读取、写入、附加等,文件类型有文本文件或二进制文件。

(4) <buffering>表示数据文件是否使用缓冲技术,其中 0 表示不缓存,1 表示只缓存一行,n 表示缓存 n 行。如果不提供或为负数,则代表使用系统隐含的缓存机制。

(5) <mode>表示的文件操作和类型，如表 7-2 所示。

表 7-2　文件操作和类型

序号	操作符	说　　明
1	w	以写入模式打开文本文件
2	r	以读取模式打开文本文件
3	a	以附加模式打开文本文件，若没有文件则创建新文件
4	w+	以读写模式打开文本文件
5	r+	以读写模式打开文本文件
6	a+	以读写模式打开文本文件
7	rb	以读取模式打开二进制文件
8	wb	以写入模式打开二进制文件
9	ab	以附加模式打开二进制文件
10	rb+	以读写模式打开二进制文件
11	wb+	以读写模式打开二进制文件
12	ab+	以读写模式打开二进制文件

说明：

(1) 以 r 模式打开的文件只能从该文件读取数据。若该文件已经存在并有数据，则系统将从该文件开始位置顺序读取数据；若打开的文件不存在，则系统会产生异常。

(2) 以 w 模式打开的文件只能向该文件写入数据。若打开的文件不存在，则以指定文件名创建该文件；若打开的文件已经存在，则将该文件原来内容覆盖后，从开始位置向该文件写入数据。

(3) 向文件写入数据的另一种方式是向一个已有文件追加数据，这时只能以 a 模式打开指定文件，若没有文件，则系统会自动创建新文件。

(4) 对于二进制文件，必须使用 b 模式表示文件类型，若是文本文件则不能使用 b 模式打开文件。

【例 7-1】 以读取模式打开文件。

在 IDLE 交互环境中，输入如下命令：

```
>>> f=open("D:\\Python36\\ch7\\f7_01.txt", "r")
```

运行结果如图 7-3 所示。

```
>>> f=open("D:\\Python36\\ch7\\f7_01.txt","r")
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    f=open("D:\\Python36\\ch7\\f7_01.txt","r")
FileNotFoundError: [Errno 2] No such file or directory: 'D:\\
Python36\\ch7\\f7_01.txt'
>>> f=open("D:\\Python36\\ch7\\f7_01.txt","r")
>>>
```

图 7-3　例 7-1 的运行结果

由于在第一次输入 open()语句时没有 f7_01.txt 文件，所以系统显示 FileNotFoundError（没有文件）异常；第二次输入 open()语句前，用记事本工具建立所需文件，如图 7-4 所示。

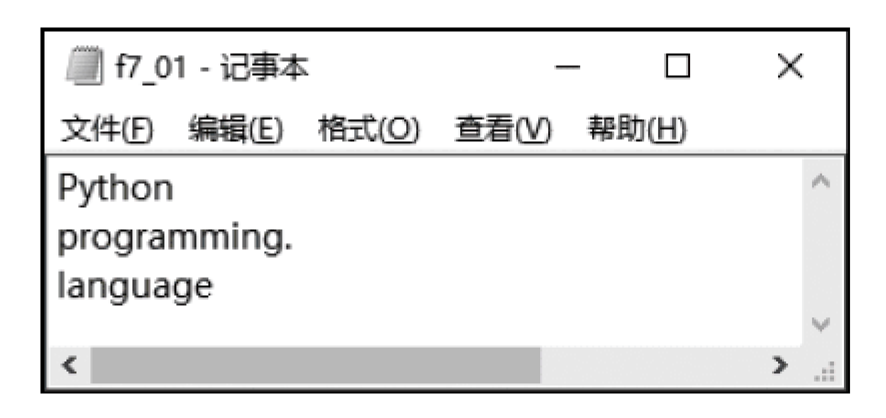


图 7-4　文件 f1_01.txt

这时，系统将能够打开文件并新建文件对象 f，f 将与 f7_01.txt 文件建立起一一对应的关系。

7.2.2　关闭文件

在计算机中，关闭一个文件具有两个含义。一是将指定文件从内存中释放，二是将指定文件存放到磁盘中，以避免数据丢失。close()函数用于关闭一个文件，即结束对指定文件的访问，删除文件对象 f。一般调用形式如下：

```
f.close()
```

这里的 f 是一个文件对象，用于指定要关闭的文件。正常完成关闭文件操作时，close()函数返回值为 True，否则返回值为 False，表示关闭文件失败。

【例 7-2】　关闭文件示例。

在 IDLE 交互环境中，输入如下命令：

```
>>>f=open("D:\\Python36\\ch7\\f7_01.txt", "r")
>>>f.close()
```

7.3　读写文本文件

Python 语言中的文件操作均是通过函数调用实现的。本节将介绍读取文本文件的函数，例如 read、readline 和 readlines，写入文本文件的 white()函数，以及辅助操作函数 seek()。

7.3.1　读取文件函数

由于 Python 没有文件操作语句，只能通过调用函数来实现对文件的访问，如表 7-3 所示。

表 7-3　读取文件函数

函　　数	说　　明
f.read([＜size＞])	隐含返回整个文件内容，若＜size＞大于 2，则内存出错，读到文件尾时返回空串
f.readline([＜size＞])	隐含读取整个文件

续表

函　　数	说　　明
f. readlines([<size>])	隐含读取整个文件并放入列表
f. tell()	返回整数，表示当前文件指针位置（到文件头的比特数）
f. seek(<offset>,[<whence>])	移动文件指针

说明：

(1) <size>：每次读取字符的个数。

(2) <offset>：表示偏移量，单位是比特，正负整数均可。

(3) <whence>：表示起始位置，值为0，表示文件头；值1，表示当前位置（隐含值）；值2，表示文件尾。

7.3.2 读取文本文件

编程时，可以通过文件对象调用多种函数来读取文件内容。

说明：

(1) 读取文件必须是以读取模式打开的。

(2) 读取的字符串可以不送入字符串变量。

(3) 读取文件时有一个位置指针，用来指向文件的当前读取字节。在文件打开时，该指针将指向文件的第一个字节。一次读取后，位置指针将自动向后移动一个位置，这样可连续多次读取，直到读取数据完毕。

1. 使用 read()函数读取文件

关于 read()函数的使用说明如下：

(1) 读取整个文件，可将读取内容存放到一个字符串变量中。

(2) 若文件大小超过可用内存空间，则不能读取文件内容。

【例 7-3】 调用 read()函数读取文件，并显示前两行。

求解方法：首先调用 read()函数读取整个文件并送入字符串变量 line 中，然后逐个检查 line 中的字符。所谓显示前两行，可由程序控制找到两个换行符为止。

源程序如下：

```
#以读取模式打开输入文件
f=open("D:\\Python36\\ch7\\f7_01.txt","r")
line=f.read()                                      #读取整个文件并送入 line 中
f.close()
#初始化三个变量
k=0
n=len(line)
num=1
print("第",num,"行: ",end="")
#控制循环两次实现显示前两行
while num<3 and k<n:
    if line[k]!="\n":                              #测试换行符
        print(line[k],end="")
```

```
            k=k+1
        else:
            num=num+1
            if num<3:
                print("\n第",num,"行: ",end="")
            k=k+1
```

运行结果如图 7-5 所示。

```
======================= RESTART: D:/Python36/ch7/p1.py =======================
第 1 行: Python
第 2 行: programming.
```

图 7-5　例 7-3 的运行结果

【例 7-4】　调用 read()函数读取文件中的所有行并显示。

求解方法：在打开文件后，使用 while 循环结构逐个读取文件中的所有行，最后要关闭文件。在 Python 中，表示文件结束就是一个空串。

在运行该程序前，可用记事本软件输入文件 f7_02. txt 中的内容，如图 7-6 所示。

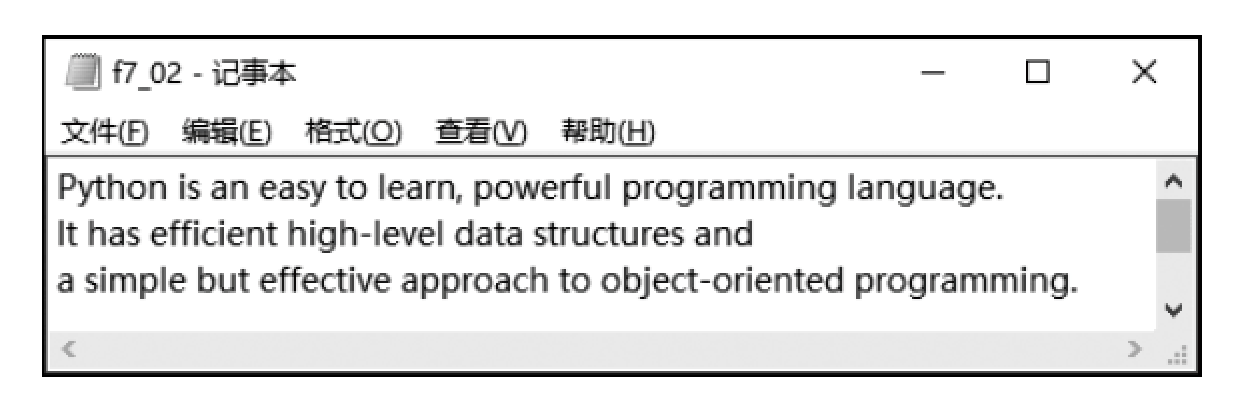


图 7-6　文件 f7_02. txt

源程序如下：

```
f=open("D:\\Python36\\ch7\\f7_02.txt","r")
line=f.read()
f.close()
print(line)
```

运行结果如图 7-7 所示。

```
======================= RESTART: D:\Python36\ch7\p1.py =======================
Python is an easy to learn, powerful programming language.
It has efficient high-level data structures and
a simple but effective approach to object-oriented programming.
```

图 7-7　例 7-4 的运行结果

【例 7-5】　调用含参数的 read()函数读取文件中的所有行。

在运行该程序前，可用记事本软件输入文件 f7_03. txt 中的内容，如图 7-8 所示。



图 7-8　文件 f7_03. txt

求解方法：read()函数可以按指定长度读取文件中的内容，若正常读取则返回字符串，否则返回 False，这可以作为循环控制条件。

源程序如下：

```
f=open("D:\\Python36\\ch7\\f7_03.txt","r")
str=""
#使用无穷循环
while True:
    #每次读取 2 个字符
    two_char=f.read(2)
    #若读取完所有字符, 则退出循环
    if not two_char:break
    #将全部读取字符连接成一个(含换行符)字符串
    str=str+two_char
f.close()
print(str)
```

运行结果如图 7-9 所示。

```
======================= RESTART: D:/Python36/ch7/p1.py =======================
Python's elegant syntax and dynamic typing,
together with its interpreted nature,
make it an ideal language for scripting and
rapid application development in many areas on most platforms.
```

图 7-9　例 7-5 的运行结果

2. 使用 readline()函数读取文件

关于 readline()函数的使用说明如下：

(1) 每次只能读取一行。

(2) 返回一个字符串对象，保存当前行的内容。

【例 7-6】 调用 readline()函数读取文件中的所有行。

源程序如下：

```
f=open("D:\\Python36\\ch7\\f7_03.txt","r")
str=""
while True:
    line=f.readline()
    if line=="":                  #读取空行表示文件结束
        break
    str=str+line                  #字符串连接运算, 串中含换行符
f.close()
print(str)
```

运行结果与上一个程序相同。

3. 使用 readlines()函数读取文件

关于 readlines()函数的使用说明如下：

(1) 一次性读取整个文件。

(2) 自动将文件内容分析成一个由若干行组成的列表。

【例 7-7】 调用 readlines ()函数读取文件中的所有行(以字符串列表形式实现)。

源程序如下:

```
f=open("D:\\Python36\\ch7\\f7_01.txt","r")
str=f.readlines()
f.close()
#以列表形式输出
print("输出列表:",str)
print("分行输出:",)
#以列表元素形式输出
for line in str:
        print("\t",line,end="")
```

运行结果如图 7-10 所示。

```
====================== RESTART: D:/Python36/ch7/p1.py ======================
输出列表: ['Python\n', 'programming. \n', 'language']
分行输出:
         Python
         programming.
         language
```

图 7-10 例 7-7 的运行结果

从运行结果可以发现,以列表元素形式输出字符串时将没有界定符(引号)。

【例 7-8】 读取文件并放入列表中。

源程序如下:

```
f=open("D:\\Python36\\ch7\\f7_01.txt","r")
my_list=f.readlines()
for i in range(0,len(my_list),1):
    print(my_list[i],end="")
```

本例中的数据文件由 3 行字符串组成,列表 my_list 就包含这 3 个元素,第 3～4 行的循环实现分 3 次输出列表 my_list 中的元素。

运行结果如图 7-11 所示。

```
====================== RESTART: D:\Python363\ch8\p1.py ======================
Python
programming
Language
```

图 7-11 例 7-8 的运行结果

4. 综合程序

【例 7-9】 分别在两个文件中保存 ABCDEFG 和 1234567,从中逐个读取字符并交叉合并成 A1B2C3D4E5F6G7。

源程序如下:

```
#打开第一个文件并读取
f1=open("D:\\Python36\\ch7\\f7_04.txt","r")
s1=f1.read()
f1.close()
```

```
#打开第二个文件并读取
f2=open("D:\\Python36\\ch7\\f7_05.txt","r")
s2=f2.read()
f2.close()
print(s1,s2)
#合成第 3 个字符串
len1=len(s1)
len2=len(s2)
s3=""
i=0
#进行交叉合并
while i<len1 and i<len2:
    s3=s3+s1[i]+s2[i]
    i=i+1
#若 s1 串较长则附加在 s3 串的末尾
while i<len1:
    s3=s3+s1[i]
    i=i+1
#若 s2 串较长则附加在 s3 串的末尾
while i<len2:
    s3=s3+s2[i]
    i=i+1
print(s3)                        #输出结果
```

运行结果如图 7-12 所示。

```
======================= RESTART: D:/Python36/ch7/p1.py =======================
ABCDEFG 1234567
A1B2C3D4E5F6G7
```

图 7-12　例 7-9 的运行结果

7.3.3　写入文本文件

写入文件与读取文件在创建文件对象上是相似的，都需要建立文件对象与相关文件的联系。所不同的是，在写入文本文件前需要以“写入”模式或“附加”模式打开文件。如果指定文件不存在，则 Python 将自动创建该文件。

与读取文件时不能添加或修改数据的限制类似，写入文件时也有不允许同时读取数据。“写入”模式打开已有文件时，会覆盖文件原有内容，从头开始，就像用一个新值覆盖写入一个变量一样。

1. 使用 write()函数写入文件

write()函数可将任何字符串写入到一个已打开的文件中，并且字符串可以是文本数据，也可以是二进制数据。它的一般调用形式如下：

```
f.write(<string>)
```

这里的参数<string>表示要写入的字符串。

注意：write()函数不会在字符串的结尾处自动添加换行符。

【例 7-10】 write()函数将字符串参数写入文件。

无论读或写文件，Python 都会跟踪文件中的读写位置。在隐含情况下，文件的读写都从文件的开始位置进行。Python 提供了控制文件读写起始位置的函数，用以改变文件读写操作发生的位置。

源程序如下：

```
#以写入模式打开文件
f=open("D:\\Python36\\ch7\\f7_06.txt","w")
f.write("Python\n")
f.write("programming\n")
f.write("language\n")
f.close()
```

本例中的第 2～4 行分别将 3 个字符串(参数)写入文件中，同时使用 close()函数关闭文件。若要验证程序是否运行成功，可使用记事本工具查看文件内容。

【例 7-11】 调用 write()函数生成一个含 3 行字符串的文件。

源程序如下：

```
f=open("D:\\Python36\\ch7\\f7_07.txt","w")
f.write("Computing Thinking\nPython Programming\n")
f.close()
f=open("D:\\Python36\\ch7\\f7_07.txt","a")
f.write("Tsinghua University Press")
f.close()
f=open("D:\\Python36\\ch7\\f7_07.txt","r")
str=f.read()
f.close()
print(str)
```

本例中的第 2 行将两个字符串写入文件，第 4 行以附加模式重新打开文件，第 5 行将一个字符串附加在文件末尾，第 7 行以读取模式第 3 次打开文件；第 8 行读取整个文件，第 10 行实现显示。

运行结果如图 7-13 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
Computing Thinking
Python Programming
Tsinghua University Press
```

图 7-13　例 7-11 的运行结果

【例 7-12】 文件复制。

源程序如下：

```
#以读取模式打开文件
f1=open("D:\\Python36\\ch7\\f7_07.txt","r")
#以写入模式打开文件
f2=open("D:\\Python36\\ch7\\f7_08.txt","w")
```

```
while True:
    str=f1.read()               #读取整个文件
    if str=="" : break
    f2.write(str)               #写入整个文件
f1.close()
f2.close()
```

本例中的循环只执行一次，由于第二次读取的将是空串，则由 break 语句退出循环。

2. 调用 seek()函数定位写入文件

seek()函数可将文件指针从当前位置移动指定位置，它的一般引用格式如下：

```
seek(<offset>[, <whence>])
```

说明：

(1) <offset>表示偏移量，单位是比特，正负整数均可，为正数表示前移，为负数表示后退；

(2) <whence>表示起始位置，为 0 表示文件首部，为 1 表示当前位置，为 2 表示文件末尾。

【例 7-13】 调用 seek()函数实现定位写入文件。

源程序如下：

```
f=open("D:\\Python36\\ch7\\f7_09.txt","w")
f.write("Python Programming")
#重新设置文件指针为 0, 即指向第 1 个字符 P
f.seek(0)
#覆盖重写前 6 个字符, 即将"Python"改写为"Pascal"
f.write("Pascal")
f.close()
f=open("D:\\Python36\\ch7\\f7_09.txt","r")
str=f.read()
print("输出结果: \n\t",str)
f.close()
```

运行结果如图 7-14 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
输出结果:
	 Pascal Programming
```

图 7-14　例 7-13 的运行结果

7.4　读写二进制文件

众所周知，计算机信息是由二进制编码表示的，因而所有文件也是由二进制的 0、1 构成的。但文件的区分是基于编码方式的，这样就有了文本文件与二进制文件的不同。为此，本书将二进制文件的扩展名设定成.bin，以区别于.txt 的文本文件。

由于 Python 没有二进制数据类型，即使位运算也是在整数类型上进行的，所以只有使

用字符串类型并以字节为单位来表示二进制数据。

7.4.1　将字符串转换为字节数据

要将字符串转换为字节数据，可以使用如下的 encode()函数：

```
str.encode(<encoding>="UTF-8", <errors>="strict")
```

说明：

(1) ＜encoding＞：表示所用的编码模式，常用的是 UTF-8 编码。

(2) ＜errors＞：设置错误处理方法，隐含 strict 表示的是，若编码出错则产生 UnicodeError 异常。

【例 7-14】 将字符串转换为字节数据示例。

在 IDLE 交互环境中，输入如下命令：

```
>>>s="Python Programming"
>>>my_bytes=s.encode("UTF-8")
>>>my_bytes
```

运行结果如图 7-15 所示。

```
>>> s="Python Programming"
>>> my_bytes=s.encode("utf-8")
>>> my_bytes
b'Python Programming'
```

图 7-15　例 7-14 的运行结果

本例中的显示信息是可识别的，但有字节数据的标识符号：b'。

7.4.2　将字节数据转换为字符串

要将字节数据转换为字符串，可以使用如下的 decode()函数：

```
str..decode(<encoding>="UTF-8", <errors>="strict")
```

由于 decode()和 encode()是一组功能相反的函数，所以参数相同。

函数以指定的编码格式解码字符串，隐含为字符串编码。

【例 7-15】 将字节数据转换为字符串示例。

在 IDLE 交互环境中，输入如下命令：

```
>>>my_bytes=b'Python Programming'
>>>s=my_bytes.decode(("UTF-8")
```

运行结果如图 7-16 所示。

```
>>> my_bytes=b'Python Programming'
>>> s=my_bytes.decode("utf-8")
>>> s
'Python Programming'
```

图 7-16　例 7-15 的运行结果

7.4.3 读写二进制文件

【例 7-16】 将 3 个学生数据写入二进制文件并读出显示。

源程序如下：

```
#声明 3 个字符串变量并初始化
s1="2017141400101: 王芳: 68: 80: 86\n"
s2="2017141400102: 陈敏: 76: 78: 90\n"
s3="2017141400103: 刘刚: 82: 82: 96\n"
#字符串转换成字节数据
my_byte1=s1.encode("UTF-8")
my_byte2=s2.encode("UTF-8")
my_byte3=s3.encode("UTF-8")
#以二进制写入模式打开文件
f=open("D:\\Python36\\ch7\\f7_10.bin","wb")
#将字节数据写入文件
f.write(my_byte1)
f.write(my_byte2)
f.write(my_byte3)
f.close()
#以二进制读取模式打开文件
f=open("D:\\Python36\\ch7\\f7_10. bin","rb")
#读取字节数据
my_1=f.readline()
my_2=f.readline()
my_3=f.readline()
#字节数据转换成字符串并显示
print(my_1.decode("UTF-8"),end="")
print(my_2.decode("UTF-8"),end="")
print(my_3.decode("UTF-8"),end="")
```

运行结果如图 7-17 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
2017141400101: 王芳: 68: 80: 86
2017141400102: 陈敏: 76: 78: 90
2017141400103: 刘刚: 82: 82: 96
```

图 7-17 例 7-16 的运行结果

注意：二进制文件 f7_09. bin 是无法用记事本工具打开的。

7.5 struct 模块

使用 struct 模块里面的 pack()、unpack()和 calcsize()函数也可以读取二进制文件。struct 模块对应 C 语言中的结构体，是将若干个不同类型的数据结合在一起。与 C 语言不同的是，struct 模块是将结构体中的数据封装成字符串来表示的。其中，在读写二进制文件时，可以调用 pack()函数、unpack()函数和 calcsize()函数。

7.5.1 pack()、unpack()和 calcsize()函数

1. pack()函数

pack()函数是按照指定格式将数据封装成字节流式的字符串，调用格式如下：

```
pack(<fmt> ,v1, v2,…,vn)
```

说明：

(1) v_1, v_2,…,v_n：表示结构体中的全部数据。

(2) <fmt>：指定结构体中的数据格式。

struct 模块支持的格式符如表 7-4 所示。

表 7-4 struct 模块支持的格式符

格式符	数据类型	所用字节	格式符	数据类型	所用字节
X	空值	1	C	单个字符	1
b	字节型	1	B	字节型	1
?	布尔型	1	H	短整型	2
H	整型	2	i	整型	4
I	整型	4	L	长整型	8
L	长整型	8	q	长整型	8
Q	长整型	8	f	单精度浮点数	4
d	双精度浮点数	8	s	字符串	1
p	字符串	1	P	整型	4

说明：

(1) 格式符 q 和 Q 要求机器安装 64 位的 Python 版本。

(2) 格式符前可以指定一个整数，表示重复个数。

(3) 格式符 s 表示指定长度的字符串，例如 4s 表示字符串长度为 4。

(4) 格式符 p 专门表示 Pascal 体系中的字符串。

(5) 格式符 P 表示用于转换一个指针，所用字节就是机器字长，通常为 8B。

【例 7-17】 pack()函数示例。

在 IDLE 交互环境中，输入如下命令：

```
>>>import struct
>>>struct.pack("ih", 123456, 789)
```

本例中的格式符 i 描述数据 123456 为 4B 长度存储的整数，格式符 h 描述数据 789 为 2B 长度存储的整数，但以二进制模式显示后，就无法直接识别。

运行结果如图 7-18 所示。

2. unpack()函数

unpack()函数将按照指定格式解析字节流式的字符串，以元组形式返回解析出来的若

```
>>> import struct
>>> struct.pack("ih",123456,789)
b'@\xe2\x01\x00\x15\x03'
```

图 7-18　例 7-17 的运行结果

干数据,其调用格式如下:

```
unpack(<fmt>, <string>)
```

说明:

(1) <fmt>:指定结构体中的数据格式。

(2) <string>:描述需解析的若干数据。

【例 7-18】　pack()函数示例。

源程序如下:

```
#导入 struct 模块
import struct
#生成含两个分量的结构体数据
my_byte=struct.pack("ih",123456,789)
#结构体数据中的两个分量的放入元组
(a,b)=struct.unpack("ih",my_byte)
#显示元组
print("第 1 个元组元素: ",a)
print("第 2 个元组元素: ",b)
```

运行结果如图 7-19 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
第1个元组元素:  123456
第2个元组元素:  789
```

图 7-19　例 7-18 的运行结果

3. calcsize()函数

calcsize()函数可以计算指定格式所占的存储字节数,其调用格式如下:

```
calcsize(<fmt>)
```

【例 7-19】　calcsize()函数示例。

在 IDLE 交互环境中,输入如下命令:

```
>>>import struct
>>>struct.calcsize("ih")
```

运行结果:

```
6
```

第 2 行代码中的格式符 i 对应 4B,格式符 h 对应 2B,合计 6B。

7.5.2　程序示例

【例 7-20】　使用 struct 模块读写二进制文件(1)。

源程序如下：

```
import struct
#创建两个含 4 个整数分量的结构体数据
mybytes_1=struct.pack("iiii",2017,80,87,68)
mybytes_2=struct.pack("iiii",2017,86,68,78)
#以写入模式打开二进制文件
f=open("D:\\Python36\\ch7\\f7_11.bin","wb")
#将两个结构体数据写入二进制文件
f.write(mybytes_1)
f.write(mybytes_2)
#f.close()
#以读取模式重新打开二进制文件
f=open("D:\\Python36\\ch7\\f7_11.bin","rb")
#读取两个含 4 个分量的结构体数据并存放到元组中
(a,b,c,d)=struct.unpack("iiii",f.read(4+4+4+4))
(e,f,g,h)=struct.unpack("iiii",f.read(4+4+4+4))
#f.close()
#显示结构体数据
print("第 1 个结构体数据：",a,b,c,d)
print("第 2 个结构体数据：",e,f,g,h)
```

运行结果如图 7-20 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
第1个结构体数据:  2017 80 87 68
第2个结构体数据:  2017 86 68 78
```

图 7-20　例 7-20 的运行结果

本例中的结构体数据全是由 4B 长度存储的整数，下面程序读写的是两种整数。

【例 7-21】 使用 struct 模块读写二进制文件(2)。

源程序如下：

```
import struct
#创建两个含 4 个整数分量的结构体数据
#第一个分量使用 4B 长度存储，后 3 个分量使用两字节存储
mybytes_1=struct.pack("ihhh",201700101,68,80,86)
mybytes_2=struct.pack("ihhh",201700102,76,78,90)
#以写入模式打开二进制文件
f=open("D:\\Python36\\ch7\\f7_12.bin","wb")
#将两个结构体数据写入二进制文件
f.write(mybytes_1)
f.write(mybytes_2)
f.close()
#以读取模式重新打开二进制文件
f=open("D:\\Python36\\ch7\\f7_12.bin","rb")
#读取两个含 4 个分量的结构体数据并存放到元组中
(a,b,c,d)=struct.unpack("ihhh",f.read(4+2+2+2))
(e,f,g,h)=struct.unpack("ihhh",f.read(4+2+2+2))
```

```
#f.close()
#显示结构体数据
print("第 1 个结构体数据：",a,b,c,d)
print("第 2 个结构体数据：",e,f,g,h)
```

运行结果如图 7-21 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
第1个结构体数据:  201700101 68 80 86
第2个结构体数据:  201700102 76 78 90
```

图 7-21　例 7-21 的运行结果

7.6　fileinput 模块

除使用 struct 模块读写二进制文件外，还可以使用 fileinput 模块和 codecs 模块实现文件操作，本节介绍 fileinput 模块，7.7 节介绍 codecs 模块。

7.6.1　fileinput 模块

fileinput 模块的功能是处理一个或多个文本文件，通常通过 for 循环来读取一个或多个文本文件的全部各行内容。

一般格式如下：

```
fileinput.input(<files>, <inplace>, <backup>, <bufsize>, <mode>, <openhook>)
```

说明：

(1) <files>：表示文件的路径列表，隐含是 stdin 文件，即标准输入文件。

(2) <inplace>：表示是否将输出结果写入文件，隐含为不写入。

(3) <backup>：表示备份文件的扩展名，例如.bak，若该文件的备份文件已存在则自动覆盖。

(4) <bufsize>：表示缓冲区大小，隐含为 0，若文件很大则可修改此参数。

(5) <mode>：表示读写模式，隐含为读取模式，见表 7-2。

(6) <openhook>：表示控制所打开文件，例如编码方式。

fileinput 模块的常用函数如表 7-5 所示。

表 7-5　常用函数

函　　数	说　　明	函　　数	说　　明
input()	返回能够用于 for 循环遍历的对象	isfirstline()	检查当前行是否是文件的第一行
filename()	返回当前文件的名称	isstdin()	判断最后一行是否从 stdin 中读取
lineno()	返回当前已经读取行的数量或序号	close()	关闭文件
filelineno()	返回当前读取行的行号		

7.6.2　程序示例

【例 7-22】 使用 fileinput 模块读取文件中的所有行。

源程序如下：

```
#导入 fileinput 模块
import fileinput
n=1
#在打开文件的同时就读取文件
for line in fileinput.input("D:\\Python36\\ch7\\f7_01.txt"):
    print("第",n,"行: ",line,end="")
    n=n+1
```

本例中的第 5 行重复读取数据文件中的一行，直到文件结束时为止。另外，变量 n 用于表示并控制行号并且文件操作选择为隐含的读取模式，其中运行过程中所用的数据文件与例 7-1 相同。

运行结果如图 7-22 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
第 1 行:  Python
第 2 行:  programming.
第 3 行:  language
```

图 7-22　例 7-22 的运行结果

注意：使用 fileinput 模块读取文件是以调用 input()函数实现的，没有前面介绍的"打开→读写→关闭"的过程，即没有关闭文件操作。

【例 7-23】　使用 fileinput 模块读取文件的行号和内容。

源程序如下：

```
import fileinput
for line in fileinput.input("D:\\Python36\\ch7\\f7_01.txt"):
    #调用 lineno()函数获得文件的当前行号
    num=fileinput.lineno()
    print(num,line)
```

本例中的 lineno()函数将返回读取文件的当前行号。由于在读取每行时已经内含换行符，所以调用 print()函数将产生两个换行。

运行结果如图 7-23 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
1 Python

2 programming.

3 language
```

图 7-23　例 7-23 的运行结果

【例 7-24】　使用 fileinput 模块读取文件中的第一行。

源程序如下：

```
import fileinput
for line in fileinput.input("D:\\Python36\\ch7\\f7_01.txt"):
    #调用 isfirstline()函数读取文件中的第一行
    if fileinput.isfirstline():
```

```
        print(line)
        break
print("over")
```

本例中的 for 循环只执行 1 次，由 break 由语句强制退出，最后程序显示结束信息。

运行结果如图 7-24 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
Python

over
```

图 7-24　例 7-24 的运行结果

【例 7-25】　遍历输入的多行字符串，直到输入为 0 时结束。

源程序如下：

```
import fileinput
print("请输入字符串：",end="")
for line in fileinput.input():
    #调用 rstrip()函数读取输入字符串
    line=line.rstrip()
    if line=="0":
        break
    else:
        print("输入字符串为：",line)
    print("请输入字符串：",end="")
print("最后输入的字符串没有显示")
fileinput.close()
```

本例中的 rstrip()函数将返回没有换行符的一行内容，其后才能直接判断 line＝＝"0"。若没有该函数，则条件 line＝＝"0"的判断结果为 False。其次，最后输入的字符串是 0，并没有显示。

运行结果如图 7-25 所示。

```
====================== RESTART: D:\Python36\ch7\p1.py ======================
请输入字符串：Python
输入字符串为： Python
请输入字符串：Programming
输入字符串为： Programming
请输入字符串：0
最后输入的字符串没有显示
```

图 7-25　例 7-25 的运行结果

7.7　codecs 模块

Python 内置模块 codecs 主要用于处理常用的编码系统，例如不同编码之间的相互转换。不过，在此只是利用 codecs 模块中的文件操作功能，但为通用起见建议读者使用 UTF-8 编码系统。

在调用 codecs 模块中的文件操作前，要首先打开文件，其一般调用格式如下：

```
fileobj=open(<filename>[, <mode>[, <code>]])
```

这里的<code>表示编码方式,通常使用 UTF-8 编码。

7.7.1 读取文本文件

下面将调用 codecs 模块读取文件,并按行存放到列表中。

【例 7-26】 调用 codecs 模块读取文件示例。

下面的代码读取了文件,将每一行的内容组成了一个列表。

源程序如下:

```
#导入 codecs 模块
import codecs
#以读取模式打开文件
f=codecs.open("D:\\Python36\\ch7\\f7_01.txt","r")
#读取文件并按行组成一个列表
lines=[]
for line in f:
    #将文件中的一行添加到列表末尾
    lines.append(line)
print("显示列表: ",lines)
print("列表元素: ")
for i in range(0,len(lines),1):
    print("\t",lines[i],end="")
f.close()
```

运行结果如图 7-26 所示。

```
======================== RESTART: D:\Python36\ch7\p1.py ========================
显示列表:  ['Python\n', 'programming. \n', 'language']
列表元素:
         Python
         programming.
         language
```

图 7-26 例 7-26 的运行结果

7.7.2 写入文本文件

写入文本文件是以列表数据形式写入的。

【例 7-27】 将列表写入文件。

源程序如下:

```
import codecs
list=[1,2,3,4,5,6,7]
s1=u"将列表写入文件: \r\n"        #u表示读取中文, \r\n为换行符
f=codecs.open("D:\\Python36\\ch7\\f7_13.txt","w")
f.write(s1)
f.write(str(list))
f.close()
f=codecs.open("D:\\Python36\\ch7\\f7_13.txt","r")
```

```
s2=f.read()
print(s2)
f.close()
```

运行结果如图 7-27 所示。

```
======================= RESTART: D:\Python36\ch7\p1.py =======================
将列表写入文件:

[1, 2, 3, 4, 5, 6, 7]
```

图 7-27　例 7-27 的运行结果

习　题　7

一、简答题

1. 什么是数据文件？由数据文件提供数据有何好处？
2. 什么是标准输入文件？什么是标准输出文件？
3. 什么是 ASCII 文件？什么是二进制文件？
4. 什么是顺序存取文件？什么是随机存取文件？
5. 简述打开文件的物理含义，如何打开文件？
6. 简述关闭文件的物理含义，如何关闭文件？
7. 如何读取文本文件？可以使用哪些内置函数？
8. 如何写入文本文件？可以使用哪些内置函数？
9. 如何读取二进制文件？可以使用哪些内置函数？
10. 如何写入二进制文件？可以使用哪些内置函数？
11. 如何调用函数 seek()定位写入文件。
12. 简述 struct 模块是如何进行读取和写入文件的。
13. 简述 fileinput 模块是如何进行读取和写入文件的。
14. 简述 codecs 模块是如何进行读取和写入文件的。

二、编程题

1. 将输入字符串中的字母全部转换成大写形式，并写入到文件 ex01.txt 中。

2. 将输入的 3 个字符串写入文本文件 ex02.txt，读取该文件中的字符串并显示。

3. 分别由 ex03.txt 和 ex04.txt 文件存放两个字符串，编程将字母串首尾相连成一个新的字符串并写入到 ex05.txt 文件中。

4. 已知 3 个学生，每个学生有 3 门课程成绩(整数)，从键盘输入全部数据并计算平均成绩，将课程成绩和平均成绩写入到文件 ex06.txt 中。

5. 接上题。将一个学生的 3 门课程成绩和平均成绩附加到 ex06.txt 文件中。

6. 在文件 ex07.txt 中存放中文“计算思维视角”，编程逐个读取并分 6 行显示。

7. 编程生成列表[1,1,1,1,2,4,8,16,3,9,27,81,4,16,32,64,256]，并按以下格式写入到文件 ex08.txt 中。

```
1   1   1   1
2   4   8   16
```

```
3   9  27   81
4  16  64  256
```

8. 编程分解数据(例如 3～9 这 7 个数)为两数之积,若不能分解则表示质数,并按以下格式写入到文件 ex09.txt 中。

```
3              这是质数
4   = 2 * 2
5              这是质数
6   = 3 * 2
7              这是质数
8   = 4 * 2
9   = 3 * 3
```

9. 在文件 ex10.txt 中存放如下文本,编程统计其中的字数(一个字符为一个字)。

Python 是一种易于学习、功能强大的编程语言。它具有高效的高级数据结构和面向对象编程的简单而有效的方法。Python 优雅的语法和动态类型,以及它的解释性质,使得它成为大多数平台上脚本和快速应用程序开发的理想语言。

10. 将字符串"welcome to study Python"(两个引号之间的内容)写入二进制文件 ex11.bin,读取该文件中的字符并显示。

11. 使用 struct 模块将如下学号和分数数据写入二进制文件 ex12.bin,并显示出来。

201712345	90	80	80
201712348	74	85	68
201712346	72	62	48

12. 使用 fileinput 模块读取文件中的所有行,并显示出来。(提示:用记事本工具软件建立文本文件并命名为 ex13.txt,内容使用上题。)

13. 使用 struct 模块将列表[1,4,9,16,25,36,49,64,81]写入文本文件 ex14.txt。

14. 使用 struct 模块读取文本文件 ex14.txt 并显示。

第 8 章　面向对象编程

面向对象编程(Object-Oriented Programming,OOP)是有别于面向过程的编程模式。从构成结构上看,对象是程序的基本单元,也是程序和数据封装的结果,从而能够提高软件的重用性、灵活性和扩展性。本章首先介绍面向对象编程的基本概念,例如对象、类、继承、多态等,然后介绍类的定义和引用,最后介绍继承、多态和重载的实现方法。

8.1　面向对象编程基础

由于对象是程序和数据封装的结果,这是与二者分离(面向过程编程)完全不同的程序构造模式。下面将介绍面向对象编程的基本概念。

8.1.1　对象与类

在进行软件开发时,一方面关心的是“如何做(How to do)”的功能实现,另一方面还要关心“做什么(What to do)”的功能应用,而使用 Python 语言中的面向对象编程技术是解决软件开发问题的最佳途径。它是通过增加软件的扩充性和重用性来提高程序员的编程效率,它的最大优点是软件具有可重用性,而且程序设计技术更接近人的思维活动。

1. 对象

在现实世界中,对象就是人类认识世界的基本单元,它可以是人,也可以是物,还可以是一件事。整个现实世界就是由各种各样的“对象”构成的,对象既可以很简单,也可以很复杂,只不过一个复杂对象是由若干个简单对象构成的。例如一个苹果、一辆汽车、一个足球、一名学生、一次游行等都可以看成是一个对象。通常,一个对象是由对象状态和对象操作两部分构成的。

(1) 对象状态。所谓对象状态就是全部静态属性的集合。如果将对象看成是一个具有状态和行为的实体,那么属于同一个类中的对象应该具有完全相同的行为,但是却可以有各自独立的状态。在这里程序员将一个对象能独立存在的原因看成是它们各自的特征,这些特征就是对象的状态,例如“学生”对象中的学号、姓名、所学专业等均是对象状态。

(2) 对象操作。在面向对象编程的方法中,对象是类的实例,类想要做的任何操作都必须通过建立对象,以及在对象上进行操作来实现。创建类中对象的过程也就是对象的实例化,从而使对象知道可以进行什么操作,以及如何修改、操作、维护定义在该对象上的数据。

2. 类

在现实世界中,“类”就是对一组具有共同属性特征和行为特征的对象所进行的抽象,类和对象之间的关系是抽象和具体的关系。类是对多个对象进行综合抽象的结果,对象又是类的个体实现,或者说一个对象就是类的一个实例。例如,由若干个的苹果可以构成苹果类,而一个苹果只是苹果类的一个对象实例。

8.1.2 对象特征

对象的主要特征包括数据抽象、封装性、模块性和易维护性。

1. 数据抽象

在现实世界中广泛存在着抽象和封装的两个概念，尤其是在科学技术飞速发展的今天，被人们广泛使用的许多家用电器，都充分体现了抽象与封装的概念，这是因为用户只关心使用效果，不关心工作原理。此外，读者在使用 Python 系统的过程中，只要能够正确地操作 Python 系统，而没有必要关心 Python 系统的实现细节，那是软件公司关注的事情。

2. 封装性

在面向对象编程技术中，程序员将数据结构和作用于数据结构上的操作组合成一个整体，数据结构的表示方式和对数据实施的操作细节都被隐藏起来，用户只能通过操作接口对数据进行操作即可。对于用户而言，只要知道如何通过操作接口对该数据进行操作，而用不着知道操作是如何实现的，也用不着知道数据是如何表示的。这就是数据封装的原理。

在面向对象编程系统中，重要的是先抽象后封装。封装是将数据抽象的外部接口与内部的实现细节清楚地分离开，即隐藏抽象的内部实现细节。抽象和封装的关系是互补的，好的抽象有利于封装，封装的实现可以进一步维护抽象的完整性。

封装是面向对象编程技术的一个重要的设计原则，也就是将对象中的各种属性和方法组合起来，并提供给外部使用者进行访问，从而保证使用者不会因为错误操作而影响对象、甚至整个程序的操作过程。另外，如果使用者想要对某些对象加以修改，则只要保证这些对象的外部使用方法和功能不发生改变，那么使用这些基本对象的程序功能也不会发生任何改变。

对象将私有元素和实现操作的内部细节隐藏后，外部程序只能通过对象所表示的具体属性、方法和外部接口等来使用对象，通过向对象发送消息来激活对象的操作。

3. 模块性

一个对象就是一个独立存在的实体，由数据结构和作用于数据结构上的操作组合而成。一个对象类似于一个黑箱，用户只是通过外部接口才能知道黑箱的功能，而内部的状态，以及如何实现这些功能的细节都是不可见的。模块化保证程序的独立性和完整性，并实现了代码的可重用性。

4. 易维护性

任何一个对象都将属性及其功能的实现细节隐藏在对象的内部。因此无论是完善对象的功能还是改变功能实现的细节，操作都被封装在对象的内部而不会影响其他对象，这大大增强对象和系统的可维护性。

8.1.3 继承

从一种对象类型派生为另外一种对象类型的主要方法是继承。这样，子对象就可以继承父对象中所有已经定义好的属性和方法而不必重新进行定义。如果子对象有自己独有的属性和方法，可以在继承后重新进行定义，但这些定义通常只是局部修改。通过重新定义子对象，可以使子对象既拥有一部分父对象的内容，又拥有自己特有的内容。对象在模拟客观世界的结构时也有层次不同，因此每个具体的对象在它所属的某一类对象的层次结构中占

一定的位置，从而具有上一层次对象的某些属性，这就是继承。

1. 继承概念

在现实生活中，继承是一个很容易理解的概念。例如，每一个人都从父母身上继承了一些特性，例如肤色、血型、性格、智力、身高等。在面向对象编程技术中，继承所描述的是对象与类之间相关的关系。这种关系使得某一类的对象可以继承另一个类中的对象。在面向对象编程中，提供继承机制是基于如下两个理由。

(1) 通过增强继承机制来减少软件模块之间的接口描述和联系实现。

(2) 继承机制充分利用软件的可重用性，避免公用代码的重复开发。

另一方面，如果没有继承机制，每次软件开发都必须“从无到有”地进行，这是因为程序员在构造各种类时，使类与类之间没有关联。继承机制使程序不再是毫无联系的类组合，而是具有良好的关联性。

2. 继承分类

从继承源上分类，继承可以分为单继承和多继承两种。

(1) 单继承是指每个子类只能直接继承一个父类的特征。

(2) 多继承是指由多个父类派生出一个子类的继承关系，多继承后的子类直接继承了多个父类的特征。

注意： Python 语言即能实现单继承，又能直接实现多继承。

8.1.4 多态性与重载

1. 多态性

在使用基本对象类型以及各种继承的对象类型过程中，如何管理这些对象所拥有的各种方法是一个非常重要的问题。在面向过程编程语言中，即使这些计算方法的功能是相似甚至是相同的，一般不允许使用同样的名字来表示不同的计算方法，因为这样计算机就不能辨认究竟程序要求的是哪个方法。例如，一个加法运算可以分为整数加法、单精度实数加法、双精度实数加法、复数加法、分数加法等。

在面向对象的程序设计技术中，使用多态性就可以解决这个问题。由于各种方法所从属的对象本身就有一定的层次关系，所以对完成同样功能的计算方法，就可以指定同样的名称。这样就极大地简化了对对象的方法调用，使用者只要记住一些基本的计算方法，剩余的工作就可依靠计算机来自动选择合适的对象，并进行正确的方法调用。

2. 重载

重载包括如下两种：方法重载和运算符重载。

方法重载是指一个标识符可重复用于多个方法的命名过程中，而运算符重载是指一个运算符可重复用于多种运算。换言之，相同名称的方法或运算符可以在不同数据中实现不同的操作。

例如，考虑对整数和实数开平方根这一操作，如果不能使用重载，程序员必须为不同类的对象声明不同的操作名称，例如 SqrtByte、SqrtInteger、SqrtLong、SqrtFloat 和 SqrtDouble。如果需要开平方根方法的对象非常多，程序员将需要记忆很多不同的方法名。在使用方法重载后，就像读者前面已经定义的类一样，读者只需记忆一个名称 Sqrt 就可以了。在发送消息时只要给出对象的数据类型信息，系统就能够确定并执行唯一正确的方法。

8.2 类的定义和引用

面向对象编程思想是将客观事物都作为实体，而对象则通过实体抽象得到。程序是通过定义一个类，对类进行实例化(也称为创建对象)来实现的。类是由类变量和类方法两部分构成的一个集合体，可以进行嵌套定义，并且类还是 Python 程序中的基本程序单位。

8.2.1 类的构成

使用 Python 编程时自然会用到对象类型机制，即自定义对象类型(即类)。类是程序的基本结构单位，由类变量和类方法两部分构成。实例化一个类，就能得到一个对象。类的类变量可以是基本类型数据、构造数据类型、类的实例等。类方法只能在类中进行定义，用来处理该类中的数据。类提供外界访问其类方法的权限，通常，类成员数据都是私有的，而类方法是公有的，外界只能访问类中的类方法，不能访问类中的私有成员数据。

8.2.2 类的定义与引用

1. 类的定义

Python 支持用 class 语句来创建类，class 关键字之后是类名，其后用一个冒号来引出类的内部定义(即类体)。用 class 语句创建类的一般格式如下：

```
class <类名>([<父类 1>, <父类 2>, …]):
    [<类变量名 1>=<初始值>]
    [<类变量名 2>=<初始值>]
    …
    [def <方法名 1>(self, 参数)]
    [def <方法名 2>(self, 参数)]
    …
```

说明：

(1) ＜类名＞：首字母最好大写，虽然语法上并无此要求，而是 Python 的编码风格。

(2) ＜类变量名＞＝＜初始值＞：定义类变量(属性)的初始值。

(3) def ＜方法名＞(self,＜参数＞)：定义类方法，其中 self 是必备参数，＜参数＞有无由方法定义的具体要求确定。

2. 类的引用

在创建类完成后，就可能通过类的实例来访问。创建对象实例的一般格式如下：

```
<实例名>=<类名>
```

可以通过对象实例来引用类变量和类方法，一般格式有两种。

格式 1：

```
<实例>.<类方法>(<参数>)
```

格式 2：

```
<实例>.<类变量>
```

这里的圆点运算符表示访问过程的层次关系。

【例 8-1】 类的定义与引用示例。

源程序如下：

```
#定义类 Student_class
class Student_class:
    #定义 3 个类变量
    num=2017081248
    name="LiBing"
    sex="男"
    #定义一个类方法
    def info(self):
        print("\n计算机学院",end="\t")
        print("计算机专业",end="\t")
        print("2017级本科生")
#主程序及其引用类
#创建 Student_cclass 类的一个实例 stud
stud=Student_class()
#通过对象访问类变量
print("学号：",stud.num)
print("姓名：",stud.name)
print("性别：",stud.sex)
#通过对象访问类方法
stud.info()
```

本例中的类 My_class 包含 num、name 和 sex 3 个类变量，以及一个类方法 info()。方法是函数在对象编程中的名称。主程序在创建类 Student_class 的实例 stud 并初始化后，就可以访问类变量和类方法。在这里并没有将实在参数传递给方法，这就是数据封装。换一个角度来看，类本身已经将数据和计算封装为一体。

运行结果如图 8-1 所示。

```
====================== RESTART: D:\Python36\ch8\p1.py ======================
学号： 2017081248
姓名： LiBing
性别： 男

计算机学院          计算机专业          2017级本科生
```

图 8-1　例 8-1 的运行结果

3. 参数 self

在例 8-1 中定义类方法 info()时，使用了参数 self。Python 编程规范要求，最好将参数 self 作为方法定义时的第一个形参，而不管有无其他参数。首先，参数 self 通常用来表示所创建的对象本身，在类方法中访问类变量时，要求以参数 self 作为前缀；其次，若在外部通过类名调用对象方法时，则需要使用参数 self 来实现参数传递(例如单向值传递)。但是，若在外部通过对象名调用对象方法时，则不需要传递参数 self。

【例 8-2】 参数 self 示例。

```
#定义类 Circle_class
class Circle_class:
    #定义类变量并赋初值
    radius=10
    def area(self):
        r=self.radius
        print("类变量：",r)
        print("圆面积：",3.1415926 * r * r)
#创建 Circle_class 类的实例 cir
cir=Circle_class()
#通过对象访问类变量
print(cir.radius)
#通过对象访问类方法
cir.area()
```

本例中的类 Circle_class 首先定义类变量 radius 并赋初值，其后由类方法引用（即 self.radius）用来计算圆面积并输出。主程序在创建类 Circle_class 的实例 cir 并初始化后，就可以直接访问类变量和类方法。

运行结果如图 8-2 所示。

```
======================= RESTART: D:\Python36\ch8\p1.py =======================
10
类变量：  10
圆面积：  314.15926
```

图 8-2 例 8-2 的运行结果

由于程序在 Circle_class 类定义的后面访问类方法，所以程序第 6 行需要添加参数 self 作为前缀；若不加，则系统将产生 NameError 异常，如图 8-3 所示。

```
======================= RESTART: D:\Python36\ch8\p1.py =======================
10
Traceback (most recent call last):
  File "D:\Python36\ch8\p1.py", line 14, in <module>
    cir.area()
  File "D:\Python36\ch8\p1.py", line 6, in area
    r=radius
NameError: name 'radius' is not defined
```

图 8-3 NameError 异常

从图 8-3 中可以发现，本例中的第 14 行在调用 area()方法时，并不知在何处获得类变量 radius，所以系统抛出 NameError（名称错误）异常。

注意：计算思维与人类的习惯思维不同。由于计算机的操作是严格按照时序进行的，例如本例中调用 area()方法时并没有如读者一样看见第 4 行，所以必须加前缀 self。读者再区分一下，若将“How old are you?”分别翻译成“怎么老是你？”或“你多大？”，各自表示的思维逻辑。

8.2.3 构造函数和析构函数

1. 构造函数__init__

构造函数__init__前后是两个下画线，也可以称为初始化函数。__init__是在创建一个

新对象时实现相关的初始化操作，如同声明变量的同时要进行初始化一样。若程序员没有定义构造函数，则 Python 将提供一个隐含的构造函数来实现创建对象时的初始化。

下面定义类 Books_class，并建立构造函数来实现对象的初始化操作。

【例 8-3】 构造函数__init__示例。

源程序如下：

```
#定义类 Books_class:
class Books_class:
    isbn=9787302123456
    price=38
    #定义构造函数__init__
    def __init__(self,a,b):
        #引用构造函数中的两个形参给 var1 和 var2 赋值
        self.var1=a
        self.var2=b
#创建 Books_class 类的一个实例 bk 并初始化
bk=Books_class("Python 程序设计","程序设计类教材")
#访问类变量
print("类变量：")
print("\t\t",bk.isbn,"\n\t\t",bk.price)
#引用构造函数中的两个形参及其赋值
print("构造函数中的形参：")
print("\t\t",bk.var1)
print("\t\t",bk.var2)
```

本例中的第 11 行在创建 Books_class 类的实例 bk 并初始化，即通过参数传递方式使形参 var1 和 var2 分别获得字符串“Python 程序设计”和“程序设计类教材”，其后就可以在第 17、18 行通过实例 bk 访问构造函数中的两个形参。

运行结果如图 8-4 所示。

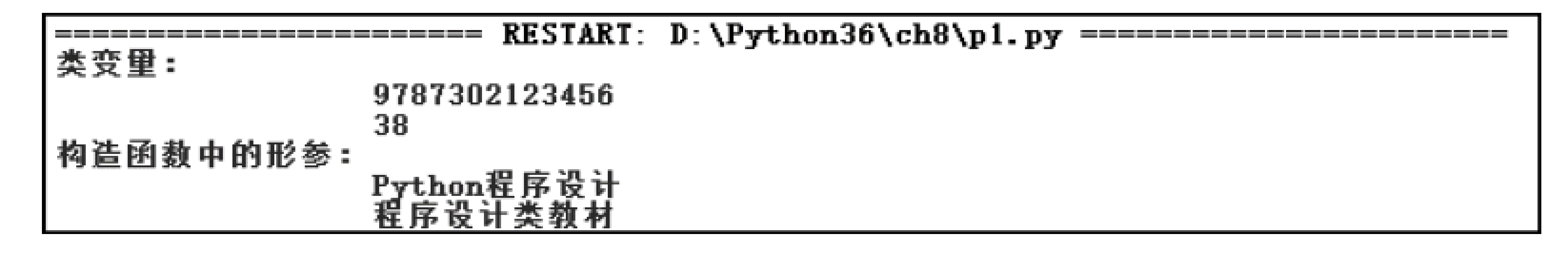


图 8-4 例 8-3 的运行结果

2. 析构函数__del__

当不使用一个对象实例时，系统应该释放所占用的各种资源，并脱离系统的管理，这时就需要使用析构函数__del__。类的析构函数__del__是与构造函数__init__正好相反的，用来释放对象占用的资源，在 Python 收回对象空间之前自动执行。如果用户没有使用析构函数，Python 将提供一个默认的析构函数进行必要的清理操作。

【例 8-4】 析构函数__del__示例。

源程序如下：

```
#定义类 My_complex
```

```
class My_complex:
    def __init__(self,realpart,imagpart):
        self.rpart=realpart
        self.ipart=imagpart
    def  __del__(self):
        print("对象 cp 已删除")
#创建 My_complex 类的一个实例 cp 并进行初始化
cp=My_complex(3,5)
#引用构造函数中的两个形参及其赋值
print("构造函数中的形参：")
print("\t\t",cp.rpart,cp.ipart)
print(cp)
#删除对象 cp
del cp
```

本例中的第 13 行试图显示实例 cp，但系统只能显示相应的内存信息，其中开头的 0x 表示 16 进制数据。第 14 行通过 del 语句调用析构函数__del__，这时系统将由析构函数__del__删除对象实例 cp，并给出提示信息。

运行结果如图 8-5 所示。

```
======================= RESTART: D:\Python36\ch8\p1.py =======================
构造函数中的形参：
		3 5
<__main__.My_complex object at 0x000000CCC24622E8>
对象cp已删除
```

图 8-5　例 8-4 的运行结果

析构函数在对象就要被垃圾回收机制处理前调用，但发生调用的具体时间是不可知的，所以建议读者最好避免在程序中使用析构函数__del__，而由 Python 解释器自行处理。

8.2.4　实例变量

在例 8-3 的构造函数中，所用参数(例如 var1)通常称为实例变量。很显然，实例变量是属于指定对象实例的，所以只能通过实例名访问；而类变量是属于类的，所以可以通过类名访问，也可以通过实例名访问，即为类的所有实例共享。

【例 8-5】 定义含有实例变量(品牌 brand，定价 price)和类变量(定价 price)的计算机类 Computer。

源程序如下：

```
#定义类 Computer
class Computer:
    #定义类变量并赋初值
    sale=3600
    #定义构造函数__init__()
    def __init__(self,str,d):
        #定义两个实例变量并赋初值
        self.brand=str
        self.price=d
```

```
    #定义 3 个类方法
    def type(self):
        print("品牌电脑")
    def display_1(self):
        print("品牌：",self.brand,"\t 定价：",self.price)
    def display_2(self):
        #类变量不能写成 self.sale，只能由类名引用
        print(Computer.sale)
#创建两个实例 pc1 和 pc2 并进行初始化
pc1=Computer("联想",3600)
pc2=Computer("戴尔",3800)
#通过实例调用类方法
pc1.type()
pc2.type()
pc1.display_1()
pc2.display_1()
#修改类变量
Computer.sale=3800
pc1.display_2()
pc2.display_2()
```

本例中两次调用无参的类方法 type()，其运行结果相同；但在修改类变量后，两次调用无参的类方法 display_2()，也将得到相同的运行结果。

运行结果如图 8-6 所示。

```
====================== RESTART: D:\Python36\ch8\p1.py ======================
品牌电脑
品牌电脑
品牌： 联想        定价： 3600
品牌： 戴尔        定价： 3800
3800
3800
```

图 8-6　例 8-5 的运行结果

8.2.5　私有成员与公有成员

Python 语言并没有对私有成员(变量)提供严格的访问保护机制。在定义类变量时，如果成员名以两个下画线“__”开头则表示是私有成员，否则是公有成员。私有成员在类的外部不能直接访问，需通过调用对象的公有类方法来访问，或者通过 Python 支持的特殊方式来访问。Python 提供访问私有成员的特殊方式，可用于测试并调试程序。

【例 8-6】 为 Handset 类定义私有成员。

源程序如下：

```
#定义类 Handset:
class Handset:
    sale=1200                   #定义类变量
    #定义构造函数__init__()
    def __init__(self,brand,price):
        self.br=brand           #定义公有成员 br
```

```
        self.__pr=price         #定义私有成员__pr
#创建实例 Hnds1 和 Hnds2 并进行初始化
Hnds1=Handset("华为销售价格：",1800)
Hnds2=Handset("小米销售价格：",1600)
#引用公有成员 br
print(Hnds1.br)
#引用私有成员__pr
print(Hnds1._Handset__pr)
print(Hnds2.br)
print(Hnds2._Handset__pr)
```

运行结果如图 8-7 所示。

```
======================= RESTART: D:\Python36\ch8\p1.py =======================
华为销售价格：
1800
小米销售价格：
1600
```

图 8-7　例 8-6 的运行结果

本例中的定义私有属性__pr，在第 14 和第 16 行访问时必须通过公有类方法实现，否则将出现 NameError 异常。下面是改写最后一行为 print(_Handset__pr)的运行结果如图 8-8 所示。

```
======================= RESTART: D:/Python36/ch8/p1.py =======================
华为销售价格：
1800
小米销售价格：
Traceback (most recent call last):
  File "D:/Python36/ch8/p1.py", line 16, in <module>
    print(_Handset__pr)
NameError: name '_Handset__pr' is not defined
```

图 8-8　例 8-6 的异常运行结果

8.2.6　公有方法与私有方法

在类中定义的方法可以粗略分为两大类，即公有方法和私有方法。公有方法与私有方法均属于对象，且私有方法的名字必须以两个下画线“__”开始。每个对象都有自己的公有方法和私有方法，这两类方法中均可以访问属于类和对象的成员。不过，公有方法是通过对象名直接调用的，私有方法则不能通过对象名直接调用，只能在属于对象的方法中通过参数 self 调用。

【例 8-7】　公有方法与私有方法示例。

源程序如下：

```
#定义类 Handset
class Handset:
    price=1000
    def __init__(self):
        #定义两个私有成员并赋初值
        self.__color="颜色：银色"
        self.__city="产地：成都"
    #定义私有方法 listColor
```

```
    def __listColor(self):
        print(self.__color)     #访问私有成员__color
    #定义私有方法 listCity
    def __listCity(self):
        print(self.__city)      #访问私有成员__city
    #定义公有方法 list
    def list(self):
        #调用两个私有方法
        self.__listColor()
        self.__listCity()
    #定义类方法 getPrice
    def getPrice():
        return Handset.price
    #定义类方法 setPrice
    def setPrice(pr):
        Handset.price=pr
#创建实例 sj 并进行初始化
sj=Handset()
#由实例名调用公有方法
sj.list()
#由类名调用类方法
print(Handset.getPrice())
Handset.setPrice(1600)
print(Handset.getPrice())
```

运行结果如图 8-9 所示。

```
====================== RESTART: D:\Python36\ch8\p1.py ======================
颜色: 银色
产地: 成都
1000
1600
```

图 8-9　例 8-7 的运行结果

8.3　继　　承

类的继承是指子类(又称为派生类)继承自父类(又称为基类)。在 Python 语言中,即可以实现单继承,又可以实现多继承。

8.3.1　单继承

在 Python 中创建类可以使用 class 语句,而创建类的继承关系也是使用 class 语句。其一般引用格式如下:

```
class <子类名>(<父类名>):
    <子类成员 1>
    <子类成员 2>
    …
```

```
    <子类方法 1>
    <子类方法 2>
    …
```

这里的<父类名>要写在括号里，以表示<子类>是继承自<父类>的。

【例 8-8】 单继承示例。

源程序如下：

```
#定义父类 IT_books
class IT_books:
    parentAttr="信息技术类图书"
    def __init__(self):
        print ("这是父类构造函数")
    def parentMethod(self):
        print("这是父类方法")
    def setAttr(self,attr):
        IT_books.parentAttr=attr
    def getAttr(self):
        print("这是属于父类的成员方法:",IT_books.parentAttr)
#定义子类 Prog_books 且继承自父类 IT_books
class Prog_books(IT_books):
    def __init__(self):
        print("这是子类构造函数")
    def childMethod(self):
        print("这是属于子类的成员方法")
#创建子类实例 exp 并进行初始化
exp=Prog_books()
#两次调用子类的方法
exp.childMethod()
exp.parentMethod()
#两次调用父类的方法
exp.setAttr(3600)
exp.getAttr()
```

运行结果如图 8-10 所示。

```
======================= RESTART: D:\Python36\ch8\p1.py =======================
这是子类构造函数
这是属于子类的成员方法
这是父类方法
这是属于父类的成员方法: 3600
```

图 8-10　例 8-8 的运行结果

8.3.2　多继承

Python 中的子类可以继承自多个父类，继承的父类列表均放在子类名后面。其一般引用格式如下：

```
#定义父类 A
```

```
class A:
…
#定义父类 B
class B:
…
#定义子类 C 继承自类父 A 和父 B
class C(A, B):
…
```

由于篇幅过大和多继承的复杂性，本书不再详细举例说明。

8.3.3 方法重写

方法重写必须出现在类的继承过程中，通常是在子类继承父类的方法后，若父类方法的功能不能满足要求，则需要对父类中的方法进行重写，即在子类中重写父类的方法。这样带来的好处是编程在继承基础上的，没有必要自行编写全部代码。

【例 8-9】 方法重写示例。

源程序如下：

```
#定义父类 PC_books
class PC_books:
    def display(self):
        print ("这是父类方法")
#定义子类 Py_book 继承自父类 PC_books
class Py_book(PC_books):
    #在子类 Py_book 中重写父类方法 display()
    def display(self):
        print("这是子类方法")
#创建子类 Py_book 的实例
bk=Py_book()
#调用由子类 Py_book 重写的方法 display()
bk.display()
```

本例中的第 3 行为定义父类方法 display()，但在第 8 行定义的子类方法 display()则是重写的，所以程序最后调用的也是重写过的子类方法 display()，并得到如下运行结果：

```
这是子类方法
```

8.4 多态与运算符重载

多态性是对相同功能的计算方法指定相同的名称，这样能够极大地简化对对象的方法调用，使用者只要记住一些基本的计算方法，剩余的工作就可依靠计算机来自动选择合适的对象，并以正确的方法调用。

8.4.1 多态

下面程序定义了一个父类 Book 及其 3 个子类，子类均将改写父类 Book 中的 display()

方法，从而使引用不同子类的同名方法而实现多态。

【例 8-10】 多态示例。

源程序如下：

```
#定义父类 Books
class Books:
    def display(self):
        #设置异常
        raise AttributeError("子类继承方法")
#定义子类 Novel 继承自父类 Books
class Novel(Books):
    def display(self):
        print("小说类图书")
#定义子类 Art 继承自父类 Books
class Art(Books):
    def display(self):
        print("艺术类图书")
#定义子类 Music 继承自父类 Books
class Music(Books):
    def display(self):
        print("音乐类图书")
#创建子类 bk1 的实例并引用多态方法
bk1=Novel()
bk1.display()
#创建子类 bk2 的实例并引用多态方法
bk2=Art()
bk2.display()
#创建子类 bk3 的实例并引用多态方法
bk3=Music()
bk3.display()
```

运行结果如图 8-11 所示。

```
====================== RESTART: D:\Python36\ch8\p1.py ======================
小说类图书
艺术类图书
音乐类图书
```

图 8-11　例 8-10 的运行结果

8.4.2　运算符重载

下面程序是对二元运算实现加、减、乘、除的运算符重载。

【例 8-11】 运算符重载示例。

```
class Operator:
    def __init__(self,a,b):
        self.a=a
        self.b=b
```

```
    #重写方法__str__(),显示 Operator 类的实例变量
    def __str__(self):
        return "Operator (%d,%d)"%(self.a,self.b)
    #重载加法运算符
    def __add__(self,other):
        x=self.a
        y=self.b
        print(x,"+",y,"=",x+y)
    #重载减法运算符
    def __sub__(self,other):
        x=self.a
        y=self.b
        print(x,"-",y,"=",x-y)
    #重载乘法运算符
    def __mul__(self,other):
        x=self.a
        y=self.b
        print(x,"*",y,"=",x*y)
    #重载除法运算符
    def __div__(self,other):
        x=self.a
        y=self.b
        print(x,"/",y,"=",x//y)
#主程序
v1=Operator(4,2)
v1.__add__("+")
v1.__sub__("-")
v1.__mul__("*")
v1.__div__("/")
```

运行结果如图 8-12 所示。

```
======================= RESTART: D:\Python36\ch8\p1.py =======================
4 + 2 = 6
4 - 2 = 2
4 * 2 = 8
4 / 2 = 2
```

图 8-12　例 8-11 的运行结果

习　题　8

一、简答题

1. 什么是对象？对象由哪两部分组成？
2. 什么是类？类由哪两部分组成？
3. 简述对象的主要特征。
4. 什么是继承？什么是单继承？什么是多继承？

5. 什么是多态？什么是运算符重载？
6. 如何定义类？如何引用类中的属性和方法？
7. 什么是构造函数？
8. 什么是析构函数？
9. 什么是方法重写？
10. 简述类变量与实例变量的异同。
11. 简述私有成员与公有成员的异同。
12. 简述公有方法和私有方法的作用。
13. 简述 Python 语言如何实现单继承。
14. 简述 Python 语言如何实现多继承。
15. 简述 Python 语言如何实现方法重写。
16. 简述 Python 语言如何实现多态。
17. 简述 Python 语言如何实现运算符重载。

二、编程题

1. 设定图书信息为 ISBN、书名、作者、出版社和定价，编程定义并访问含类变量和类方法的类，其中类方法能够显示所有图书信息。

2. 定义教师 Teacher 类，其中设定类变量 age，要求编写类方法通过参数 self 引用类变量 age，显示距 60 岁的年数。

3. 编程实现多态，分别基于一个数据计算正方形、圆形、等腰直角三角形、等边三角形的面积。

三、操作题

1. 在编程窗口中输入如下程序，并写出运行结果。

```
#定义 Books 类
class Books:
    price=32                        #类变量
    def display(self):              #类方法
        print("\tPython Programming")
        print("\t--Computation Thinking Perspective")
#创建 Books 类的一个实例 study
study=Books()
#引用实例 study 中的类方法
print("书名：")
study.display()
#引用实例 study 中的类变量
print("定价：",end="\t")
print(study.price)
```

2. 在编程窗口中输入如下程序，并写出运行结果。

```
#定义类 My_complex
class My_complex:
    #定义构造函数__init__
```

```
    def __init__(self,real,imag):
        self.re=real
        self.im=imag
#创建 My_complex 类的一个实例 x 并进行初始化
x=My_complex(3,5)
print("复数：")
print("\t 实部=",x.re,end="\t")
print("虚部=",x.im)
```

第 9 章　异 常 处 理

任何程序在运行过程中,都可能会产生错误。计算机如何处理这些错误?用什么方式来处理错误?均是程序员必须面对的问题。与大多数计算机语言一样,Python 也是采用“异常方式”来处理并解决程序错误的。本章将详细说明 Python 语言的异常处理机制,包括异常抛出和捕捉的方法,以及在 Python 程序中如何处理异常。最后,介绍如何自定义异常类。

9.1　程序错误及其处理

所谓“异常”就是计算机程序执行过程中出现不正确操作的事件。不严重时,这些异常只是产生一个错误的运算结果;严重时,这些异常将导致整个计算机系统的崩溃。严重的异常可能包括如下几类:

(1) 在进行除运算时出现除数为零的异常。

(2) 在进行开平方根时出现负数的异常。

(3) 在程序处理大量数据过程中出现内存空间不够的异常。

(4) 在对数据文件进行写盘操作时出现磁盘空间不够的异常。

(5) 在对数据文件进行读操作时出现磁盘文件没有打开的异常。

在这里重申一点,异常处理是关于程序错误的,所以原则上不能在 IDLE 交互环境中实现。这是基于单语句方式是无关上下文的,至少完全与下文(指后续语句)无关,从而使程序错误被局限在特定语句块中,所以不能称为程序错误,而是单条语句错误。

9.1.1　程序错误类型

在程序中,可能会出现如下 4 种基本错误:语法错误、输入错误、运行错误和逻辑错误。

1. 语法错误

语法错误很可能是由程序员在编程或输入程序时造成的错误,例如拼错关键字、列表名、变量名、函数名或常量,大小写字母混用,没有指定必须的分隔符号,缩进错误等。语法错误通常发生在程序员编写源程序的过程中,一般计算机语言可以在翻译过程中检查出语法错误,有许多语法错误都会自动出现在翻译时的错误信息中。

下面介绍两种语法错误:错误结束标记和翻译错误。

(1) 错误结束标记。结束标记是完整语句的一部分,若结束标记出错则语句也将出错。

【例 9-1】　错误结束标记示例。

源程序如下:

```
print("=================")
print("Python Programming")
print("=================)
```

若试图运行程序，则系统将显示错误信息，表示扫描到字符串结束标记，这时的弹出窗口如图 9-1 所示。

在图 9-1 中显示的 EOL 缩写自“End Of Line”，表示一行结束。

同时，在编辑窗口中可以发现有一个淡红色长条，以标记对应的出错代码行位置（第 3 行）。由于 print()函数的双引号（用于界定提示信息）是配对使用的，所以只要添加双引号就可以修改成功，这样的字符串才能构成正确的提示信息，如图 9-2 所示。

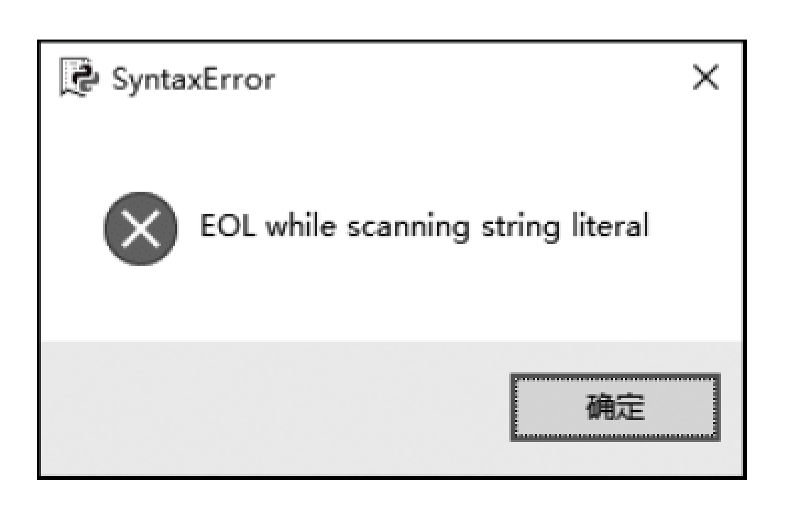


图 9-1　语法错误示例

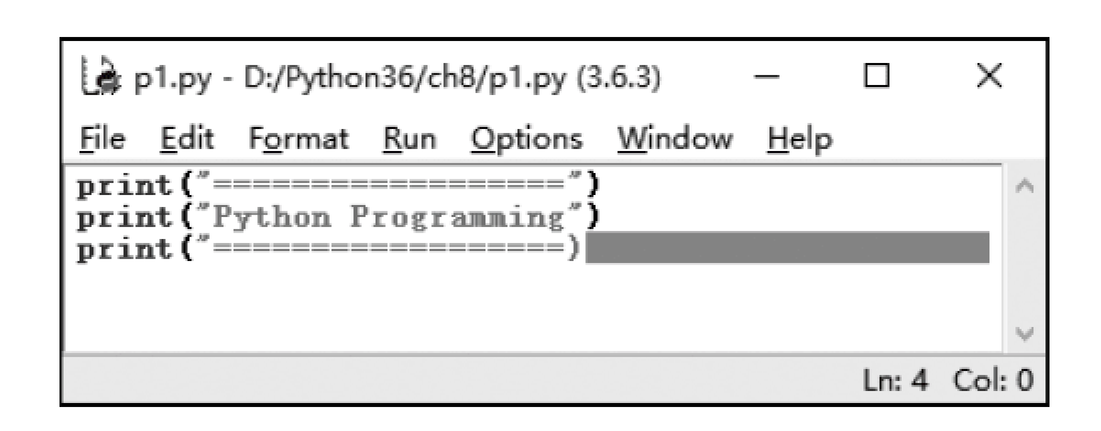


图 9-2　编辑窗口中的语法错误处理

注意：编辑窗口右下角的行、列号（第 3 行，第 26 列）是当前光标位置，即淡红色长条标记的最右侧位置，Python 解释器不会（也不可能）指定出错的精确位置。

（2）翻译错误。还有一种语法错误是翻译错误。翻译错误是指一般程序设计语言的翻译器不能正常翻译的错误。通常出现在 Python 解释器要翻译的某一程序段中，而解释器又不能正确翻译其中的某条语句时。很显然，有翻译错误的任何程序是不可能被机器执行的。

【例 9-2】　翻译错误示例。

源程序如下：

```
fact=1
for i in range(1,11)
    fact=fact*i
print("10!=",fact)
```

说明：程序中的第 2 行没有语句层次符号“:”，这属于语法错误。将该程序翻译后，得到如图 9-3 所示的显示信息。

如图 9-3 所示，invalid syntax 表示语法错误。同时，在编辑窗口中可以发现有一个淡红色长条，以标记对应的出错代码行位置（第 2 行），如图 9-4 所示。由于 for 循环需要用冒号表示下一层次的循环体，所以只要添加冒号就可修改成功。

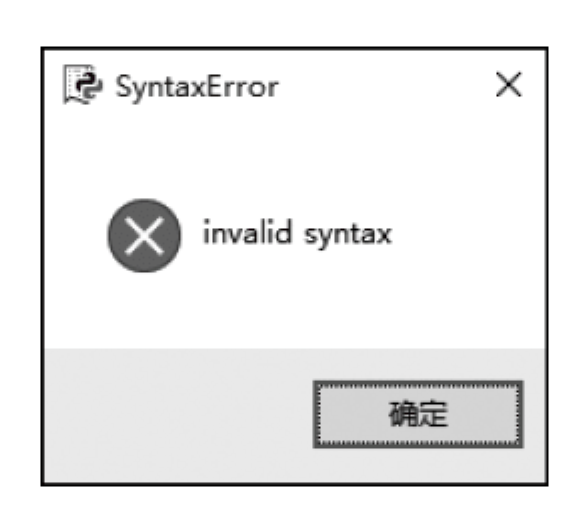


图 9-3　翻译错误示例

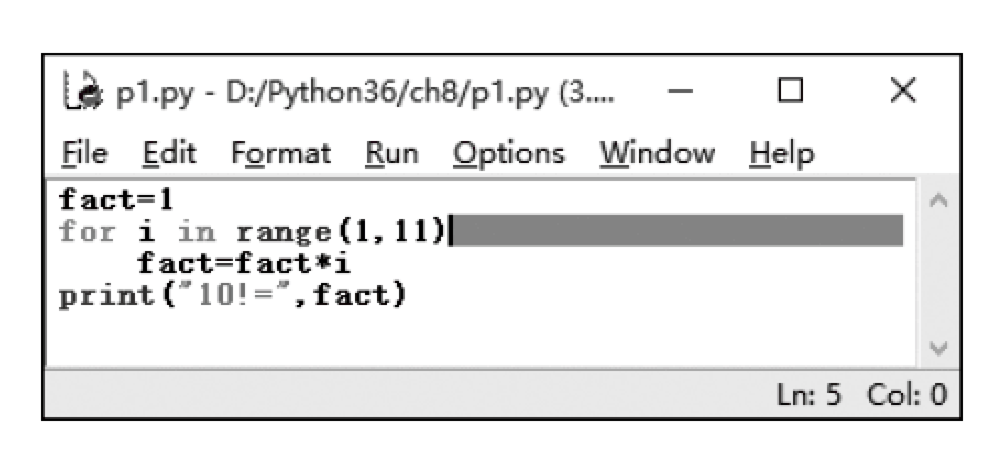


图 9-4　翻译错误示例

本例中，由于 Python 系统认定 for 循环中的循环体出现层次问题，从而导致翻译过程被终止，所以后续语句也就不会执行。其次，Python 编辑器具有智能识别，即程序中的第 3 行应该自动由编辑器产生 4 格缩进。若没有缩进则读者应该仔细检查程序，而不是强行输入 4 个空格。

2. 输入错误

程序通用性体现在输入数据的不同，但输入数据也是可能出错的。

【例 9-3】 由输入圆的半径计算面积。

源程序如下：

```
r=int(input("请输入半径: "))
area=3.14*r*r
print("圆面积: ",area)
```

若试图运行程序，则系统将显示错误信息，这时的弹出窗口如图 9-5 所示。

```
======================= RESTART: D:/Python36/ch9/p1.py =======================
请输入半径: 3.5
Traceback (most recent call last):
  File "D:/Python36/ch9/p1.py", line 1, in <module>
    r=int(input("请输入半径: "))
ValueError: invalid literal for int() with base 10: '3.5'
```

图 9-5　输入错误

从运行结果可以发现，程序需要输入一个整数作为圆的半径，这时 int()函数要求的字符串为不含小数点的整数。而当输入字符串为 3.5 后，系统将抛出 ValueError(值错误)异常。另外，由于 Python 系统认定 int()函数不能进行数据类型转换，从而导致翻译过程终止，所以后续语句也就不会执行。

3. 运行错误

运行错误是指一般程序设计语言出现了不能执行的表达式或语句，这种错误是 Python 解释器本身不能发现的。例如无效操作、无效函数、无效参数、无效数学运算等都是运行错误。运行错误发生在程序代码的运行过程中，典型的运行错误包括以下几种：表达式运算发生下溢或上溢，负数开平方根，堆栈容量不够，表达式中的数据类型不匹配，试图打开一个不存在的文件，序列索引超界，用零除一个数，等等。

【例 9-4】 运行错误示例。

源程序如下：

```
n=10
k=100/(n-10)
print("k=",k)
```

将本例中的第 3 行将出现用零除以一个数的错误，唯有运行过程才能知道分母 n－10 为 0。这时系统将抛出 ZeroDivisionError(被零除)异常，同时也就不会运行第 3 行的输出语句。由于 Python 解释过程只能对程序进行静态的语法检查，不可能检查运行出现的错误，如图 9-6 所示。

读者在面临运行错误时，只有仔细检查程序清单，以便发现错误并进行修改。

```
====================== RESTART: D:/Python36/ch9/p1.py ======================
Traceback (most recent call last):
  File "D:/Python36/ch9/p1.py", line 2, in <module>
    k=100/(n-10)
ZeroDivisionError: division by zero
```

图 9-6　运行错误示例

4. 逻辑错误

逻辑错误是由程序员自身推理错误造成的，例如将正确的表达式 100/(n－10)写成 100＊(n－10)。与运行错误一样，翻译程序并不能检查这种错误。有时，逻辑错误也会导致运行错误。常见的逻辑错误有如下几种情况。

(1) 程序设计的解题步骤与计算方法不正确。

(2) 由程序员定义的算法可能产生一个不正确的运算结果。

(3) 某个函数执行的是一个错误的计算问题。

(4) 不正确或不完全地执行一个计算问题。

(5) 程序设计的逻辑结构本身是不正确的。

注意：有些程序错误并不会导致异常。通常情况是，在 Python 解释器无法运行程序时才会发生一个异常，同时生成一个异常对象并表示特定的错误，以便进行处理。实际上，计算机作为计算工具，全部操作由程序控制，故机器不可能主动去检查程序中的错误。在 Python 中，异常只能发生在程序运行或试图运行这一前提下。换言之，若程序错误是由语句书写和求解算法导致的，则程序员应该自行检查程序，唯有计算机运行出现的异常才是需要通过编程实现的处理。

9.1.2　程序运行错误处理方法

一般程序设计语言的运行错误处理方法有两种：终止程序运行和错误代码检测。

1. 终止程序运行方法

在执行应用程序中出现一般程序设计语言的错误时，可以终止正在运行的程序，然后进行仔细检查。这种错误处理方法具有以下的优缺点。

(1) 优点：错误处理方法简单灵活，不增加计算机系统的任何负担。

(2) 缺点：给程序员增加许多负担，甚至可能无法重新运行程序，并对整个错误情况失去控制，最终使程序调试工作不能继续进行。

2. 错误代码检测方法

在执行应用程序中出现一般程序设计语言的错误时，可以终止正在运行的程序，然后引入错误代码检测机制。即在程序中利用“错误代码检测”机制对错误返回一个特定的预定义值，程序员可以根据不同的返回值进行识别以找出错误，并最终纠正错误。错误代码检测方法具有以下的优缺点。

(1) 优点：错误处理方法非常直接，容易查找到程序中的各种错误。

(2) 缺点：一方面，这种错误处理方法要增加计算机系统的负担。另一方面，错误检测的代码和正确程序的代码是混合在一起的，这样使得错误处理过程非常复杂，从而大大降低程序的可读性和可维护性。

9.2 标准异常

为方便程序员编程和调试程序，Python 提供分层次结构的许多标准异常。当然，也允许用户自行定义异常。

9.2.1 标准异常

Python 提供的常用标准异常如表 9-1 所示。

表 9-1 常用标准异常(以字母序排列)

标准异常	说　明
AssertionError	断言语句失败
AttributeError	试图访问未知对象的属性
EOFError	输入文件末尾标志 EOF(Ctrl+D)
FloatingPointError	浮点计算错误
GeneratorExit	调用 generator. close()方法
ImportError	导入模块失败
IndentationError	缩进错误
IndexError	索引超出序列范围
KeyboardInterrupt	用户输入中断键(Ctrl+C)
KeyError	字典中查找一个不存在的关键字
MemoryError	内存溢出(可通过删除对象释放内存)
NameError	试图访问一个没有的变量
NotImplementedError	没有实现的方法
OSError	操作系统产生的异常(例如打开一个没有的文件)
OverflowError	数值运算超出最大限制
ReferenceError	弱引用试图访问一个已经被垃圾回收机制回收的对象
RuntimeError	一般运行时错误
StopIteration	迭代器没有更多的值
SyntaxError	语法错误
SystemError	系统错误
SystemExit	进程被关闭
TabError	Tab 和空格混合使用
TypeError	不同类型数据间的无效操作
UnboundLocalError	访问一个未初始化的本地变量(NameError 的子类)

续表

标准异常	说　　明
UnicodeDecodeError	Unicode 解码时出错(UnicodeError 的子类)
UnicodeEncodeError	Unicode 编码时出错(UnicodeError 的子类)
UnicodeError	Unicode 相关的错误(ValueError 的子类)
UnicodeTranslateError	Unicode 转换时出错(UnicodeError 的子类)
ValueError	传入无效参数
ZeroDivisionError	除数为零

9.2.2　标准异常示例

为了较深入地理解常用标准异常,下面通过相应语句来说明 AttributeErro(属性异常)、IndexError(索引异常)、NameError(名称异常)、ValueError(值异常)、ZeroDivisionError(除数为零异常)等异常。

(1) AttributeError。在 IDLE 交互环境中,输入如下命令:

```
>>>m=1
>>>m.show()
```

运行结果如图 9-7 所示。

```
>>> m=1
>>> m.show()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    m.show()
AttributeError: 'int' object has no attribute 'show'
```

图 9-7　AttributeError 的异常运行结果

该结果表示 int 对象没有 show 属性。

(2) IndexError。在 IDLE 交互环境中,输入如下命令:

```
>>>a=[1, 2, 3, 4, 5]
>>>a[5]
```

运行结果如图 9-8 所示。

```
>>> a=[1,2,3,4,5]
>>> a[5]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a[5]
IndexError: list index out of range
```

图 9-8　IndexError 的异常运行结果

该结果表示列表 a 只有 5 个索引(0,1,2,3,4),因而在引用 a 时不能使用第 5 号索引。

(3) NameError。在 IDLE 交互环境中,输入如下命令:

```
>>>x=10
>>>x=10+y
```

运行结果如图 9-9 所示。

```
>>> x=10
>>> x+y
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    x+y
NameError: name 'y' is not defined
```

图 9-9　NameError 的异常运行结果

该结果表示变量 y 没有定义,这是因为不能访问没有名称的变量或有名称但无值的变量。

(4) ValueError。在 IDLE 交互环境中,输入如下命令:

```
>>>int("xyz")
```

运行结果如图 9-10 所示。

```
>>> int("xyz")
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    int("xyz")
ValueError: invalid literal for int() with base 10: 'xyz'
```

图 9-10　ValueError 的异常运行结果

该结果表示 int()函数只能将数字串转换为整数,所以字符串属于无效参数。

(5) ZeroDivisionError。在 IDLE 交互环境中,输入如下命令:

```
>>>10/0
```

运行结果如图 9-11 所示,图中显示除数不能为零。

```
>>> 10/0
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    10/0
ZeroDivisionError: division by zero
```

图 9-11　ZeroDivisionError 的异常运行结果

从上面示例中可以发现,所有程序错误信息均由系统通过英文进行描述。最基本的调试方法是基于代码所在的行号,通过仔细检查源程序清单中的相应语句,这样可以解决绝大多数语法错误。而对于运行错误和逻辑错误,读者只能通过较长时间的程序调试过程才能解决。

9.3　抛出异常和捕捉异常

程序运行过程中,异常可以由错误本身自动抛出,也可以由程序中的代码抛出。抛出的异常被捕捉,就从正常代码中跳出来。因此异常是可以改变程序控制流程的一种事件。在程序中,可以增加处理这些异常的方法或给出错误报告,甚至可以终止运行程序,当然异常处理并不一定意味着要终止程序。如果不是严重错误,异常处理后,程序可以从错误情况下恢复执行。

9.3.1　抛出异常

有两种方式抛出异常,一种方式是程序运行过程中由错误自动抛出异常,另一种是编程

中使用异常抛出语句 raise 或 assert 人为地抛出异常。

(1) 由系统自动抛出异常。

【例 9-5】 因数据类型不一致而自动抛出异常。

源程序如下：

```
a=10
s="20"
result=a+s
print("result=",result)
```

运行结果如图 9-12 所示。

```
======================= RESTART: D:/Python36/ch9/p1.py =======================
Traceback (most recent call last):
  File "D:/Python36/ch9/p1.py", line 3, in <module>
    result=a+s
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

图 9-12 例 9-5 的运行结果

本例中的第 3 行试图进行两数求和，显然这里的变量 s 存放内容是一个数字串而不是数字。所以系统将自动抛出 TypeError(类型错误)异常。要处理此异常，只要删除定义字符串的双引号就可修改成功。

【例 9-6】 因无法调用函数自动抛出的异常。

源程序如下：

```
import math
n=float(input("n="))
m=sqrt(n)
print(n,"的平方根是",m)
```

运行结果如图 9-13 所示。

```
======================= RESTART: D:/Python36/ch9/p1.py =======================
n=4
Traceback (most recent call last):
  File "D:/Python36/ch9/p1.py", line 3, in <module>
    m=sqrt(n)
NameError: name 'sqrt' is not defined
```

图 9-13 例 9-6 的异常运行结果

本例中的第 3 行试图进行开平方根运算，但系统将自动抛出 NameError(名称错误)异常，表示系统不能识别 sqrt()。要处理此异常，只要在 sqrt()前添加模块名 math 就可修改成功。

(2) 由 raise 语句抛出异常。下面介绍 raise 语句抛出异常的过程。

raise 语句的一般引用格式如下：

```
raise <异常类>
raise <异常类实例>
```

说明：raise 后面通常跟<异常类>或者<异常类实例>，可以是 Python 内置的标准异常，也可以是用户自定义的异常类。如果省略<异常类>或者<异常类实例>，则抛出最近发生的异常。

【例 9-7】 使用 raise 语句抛出异常。

源程序如下：

```
import math
n=int(input("n="))
if n<0 : raise ValueError("数值不能为负")
print(n,"的平方根：",math.sqrt(n))
```

运行结果如图 9-14 所示。

```
====================== RESTART: D:\Python36\ch9\p1.py ======================
n=-10
Traceback (most recent call last):
  File "D:\Python36\ch9\p1.py", line 3, in <module>
    if n<0 : raise ValueError("数值不能为负")
ValueError: 数值不能为负
>>>
====================== RESTART: D:\Python36\ch9\p1.py ======================
n=10
10 的平方根： 3.1622776601683795
```

图 9-14 例 9-7 的异常运行结果

本例中的第 2 行由 raise 语句人为抛出 ValueError(值错误)异常，并给出提示信息：数值不能为负。注意，这里的中文信息是语句生成的。这里的运行结果是运行两次程序所得，第一次有异常，第二次没有异常。

在编程中，还可以使用 assert(断言)语句抛出异常，可参阅 9.4 节。

【例 9-8】 设定直角三角形的 3 条边从小到大为 *a*、*b*、*c*，在计算面积前测试直角三角形并使用 raise 语句抛出异常。

源程序如下：

```
a=int(input("a="))
b=int(input("b="))
c=int(input("c="))
if a*a+b*b!=c*c:
    raise ValueError("这不是直角三角形")
else:
    print("直角三角形面积：",a*b/2)
```

运行结果如图 9-15 所示。

```
====================== RESTART: D:\Python36\ch9\p1.py ======================
a=3
b=5
c=4
Traceback (most recent call last):
  File "D:\Python36\ch9\p1.py", line 5, in <module>
    raise ValueError("这不是直角三角形")
ValueError: 这不是直角三角形
```

图 9-15 例 9-8 的异常运行结果

在执行程序时，若输入数据 3、4、5，则系统将执行第 7 行并输出直角三角形面积。

9.3.2 捕捉异常

在 Python 中，各种捕捉异常的方式是一致的。为了捕捉异常，常常把可能会出现异常的语句块放置在 try 语句中，由 except 子句捕捉语句块中发生的所有异常。

1. try 语句

try 语句的一般格式如下：

```
try:
    <代码或 raise/assert 语句>
except [<异常名>]:
    <处理异常的语句>
[else:
    <语句块>]
```

功能：try 语句块中捕捉的各种异常均由 except 子句分别处理。except 子句后描述该子句处理的异常名称，冒号后编写相应的处理语句。else 部分为没有异常捕捉时执行的语句块，可以省略。

说明：

(1) 每一条 try 语句至少要对应一个 except 子句。

(2) 一条 try 语句中可以放置多条可能产生异常的语句，其中任何语句都可以抛出异常而被相应的 except 子句捕捉。

(3) except 子句后带有异常名称，其中的处理语句将针对该异常进行处理。可以同时有多个异常名称，若多个异常名称放在 except 子句后面的括号中，之间要用逗号分开。

(4) 没有给出任何异常名称的 except 子句用于处理所有没有预先列出的异常。因此不带异常名称的 except 子句一般要放在带异常名称的 except 子句后面。

(5) 一个异常被一条 except 子句处理后，就不会被其他 except 子句重复处理。

(6) try 语句不仅能够捕捉异常，还能在异常得到处理后会回到 try 语句后继续运行，而不是被 Python 解释器终止。

(7) 如果产生一个异常没有被捕捉并得到处理，它将一直向上传递，最终到模块的最顶层或由 Python 隐含的异常机制来处理。

【例 9-9】 字符串异常处理。

源程序如下：

```
try:
    s="Python"
    for i in range(0,7) : print(s[i],end="")
except IndexError:
    print("\n 索引越界错误")
    print("这是 except 子句")
else:
    print("\n 这是 else 子句")
```

本例中的第 3 行试图访问 a[6]（指字符串中的第 7 个字符，但字符串的长度为 6），系统将抛出 IndexError（索引越界错误）异常；第 5 行在捕捉到该异常后，通过显示信息“索引越界错误”来处理异常；其次，第 8 行中的语句是不被执行的。

运行结果如图 9-16 所示。

若将循环终值的 7 改成 6，则程序将正常运行。

```
====================== RESTART: D:\Python36\ch9\p1.py ======================
Python
索引越界错误
这是except子句
```

图 9-16　例 9-9 的异常运行结果

【例 9-10】　文件异常处理。

源程序如下：

```
try:
    f=open("D:\\Python36\\ch9\\f9_01.txt ","r")
except IOError:
    print("没有所需文件")
else:
    print("文件正常打开")
finally:
    f.close()
```

本例中的第 2 行试图以读取方式打开一个没有的文件，系统将抛出 IOError(输入输出结束)异常；第 4 行在捕捉到该异常后，通过显示信息“没有所需文件”来处理异常；其次，第 8 行中的文件 f 是没有定义的，即对应 NameError(名称错误)异常。

运行结果如图 9-17 所示。

```
======================= RESTART: D:\Python36\ch9\p1.py =======================
没有所需文件
Traceback (most recent call last):
  File "D:\Python36\ch9\p1.py", line 8, in <module>
    f.close()
NameError: name 'f' is not defined
```

图 9-17　例 9-10 的异常运行结果

若指定文件存在，则程序能够正常运行。读者可以自行完成。

2. finally 子句与异常处理

finally 子句用于清理在 try 语句中执行的操作，例如释放所占资源(例如文件流、游戏控制、数据库连接、图形处理等)，而不用等待由运行库中的垃圾回收机制来处理这些对象。通常情况是 finally 子句在执行完 try 语句和 except 子句后才执行，与是否抛出异常或者是否找到与异常类型匹配的 except 子句无关。

【例 9-11】　finally 子句与异常处理示例。

程序要求将输入的字符串写入到文本文件中，直到输入字母 Q 时结束。如果按 Ctrl+C 组合键终止程序运行，则要保证已打开文件能够正常关闭。

源程序如下：

```
try:
    #以写入方式打开文件
    f=open("D:\\Python36\\ch9\\f9_02.txt ","w")
    #使用无穷循环
    while True:
        s=input("输入字符串(按 Q 结束): ")
        if s=="Q" : break             #输入 Q 退出循环
```

```
        f.write(s+"\n")
except KeyboardInterrupt:
    print("按 Ctrl+C 组合键中断执行程序")
finally:
    f.close()
    print("文件能够正常关闭")
```

本例中的第 3 行以写入方式打开文件 f9_02.txt 来保存输入字符串；第 4～7 行将若干字符串写入文件，直到输入 Q 时结束。运行结果如图 9-18 所示。

```
====================== RESTART: D:/Python36/ch9/p1.py ======================
输入字符串（按Q结束）: Python
输入字符串（按Q结束）: Programming
输入字符串（按Q结束）: Q
文件能够正常关闭
```

图 9-18　例 9-11 的运行结果

如果按 Ctrl＋C 组合键终止程序运行，则要保证已打开的文件能够正常关闭。从运行结果可以发现，这里的 finally 子句的执行与是否抛出异常完全无关。

以上程序结束后，就可以在指定文件夹中生成并保存输入字符串的文件 f9_02.txt。

9.4　断　　言

assert 又称为断言语句，属于有条件的抛出异常，功能及使用与 raise 语句相似。

9.4.1　断言概念

断言的功能就是帮助程序员调试程序，以便保证程序能够正常运行，例如需要通过变量的内容来判断程序运行情况，这样能够比调用 print()函数来显示结果更有效。断言一般用于下列情况：

（1）检查程序中的常量定义。

（2）编制防御性程序段。

（3）检测程序运行时的逻辑关系。

（4）检查条件设置的合法性。

（5）检查程序文档的完整性。

9.4.2　assert 语句

assert 语句与 raise 语句相似，只不过在 assert 中添加一个条件表达式，若表达式为假值，则抛出 AssertionError 异常，否则没有抛出异常。

assert 语句的一般引用格式如下：

```
assert <条件表达式>[, <参数>]
```

说明：

（1）＜条件表达式＞用于控制是否抛出 AssertionError(判断错误)异常。

（2）若指定＜参数＞，则在抛出 AssertionError 异常后将＜参数＞作为提示信息同时

抛出。

与前面的例 9-7 类似，当条件 a<0 成立时将抛出 AssertionError 异常，并将“数值不能为负”作为提示信息同时抛出。写出程序段如下：

```
>>>a=-1
>>>assert a<0, "数值不能为负"
```

【例 9-12】 断言示例：负数不能开平方根。

源程序如下：

```
import math
n=int(input("整数 n: "))
assert n<0,"平方根不能为负"
result=math.sqrt(n)
print("平方根=",result)
```

本例中的第 3 行设置断言条件是 n<0，提示信息是“平方根不能为负”。由于 Python 系统规定标准异常优先于断言设置，所以没有显示该提示信息。

运行结果如图 9-19 所示。

```
======================= RESTART: D:/Python36/ch9/p1.py =======================
整数n: -4
Traceback (most recent call last):
  File "D:/Python36/ch9/p1.py", line 4, in <module>
    result=math.sqrt(n)
ValueError: math domain error
```

图 9-19 例 9-12 的运行结果

【例 9-13】 断言示例：除数不能为 0。

源程序如下：

```
x=int(input("整数 x: "))
y=int(input("整数 y: "))
assert y!=0,"除数不能为 0"
result=x/y
print(x,"/",y,"=",result)
```

本例中的第 3 行设置断言条件是 y!=0，提示信息是“除数不能为 0”，执行第 3 行语句将抛出 AssertionError（断言错误）异常。

运行结果如图 9-20 所示。

```
======================= RESTART: D:\Python36\ch9\p1.py =======================
整数x: 2
整数y: 0
Traceback (most recent call last):
  File "D:\Python36\ch9\p1.py", line 3, in <module>
    assert y!=0,"除数不能为0"
AssertionError: 除数不能为0
```

图 9-20 例 9-13 的运行结果

综上所述，使用 assert 语句处理异常比使用 try 语句处理异常更简单。

9.5 自定义异常类

前面介绍的内容均属于内置的标准异常,下面将结合面向对象编程技术定义并访问自定义异常类。

9.5.1 引言

在 Python 中,异常是作为对象进行处理的。实际上,Python 内置的大多数异常类都是继承自父类 Exception 和 BaseException。虽然 Python 的内置异常类也可以分为父类和子类两种,但是绝大多数异常都是 Exception 类和 BaseException 类的子类。每次抛出异常时系统都将同时生成一个错误类的对象实例。另一方面,编程时除使用内置异常外,用户也可以自行定义异常类,从而设计出新的异常类,使得用户可以处理各种特定的程序错误。为了提高编程效率,定义异常类与一般类对象定义的方法和原理基本相同,只不过都将直接或间接地继承父类 Exception 和 BaseException。为了与标准异常保持一致,自定义异常类在命名时也以 Error 或 Exception 作为后缀。

9.5.2 程序示例

【例 9-14】 创建自定义异常类,用于处理程序中出现负数的异常。

源程序如下:

```
#自定义异常类 ScoreError 继承自父类 Exception 类
class ScoreError(Exception):
    #定义构造方法__init__()
    def __init__(self,score):
        Exception.__init__(self,score)
        self.score=score
    #定义方法__str__()
    def __str__(self):
        return self.score+":分数不能为负数"
#定义函数 total()
def total(score):
    total=0
    for n in score:
        #由 raise 语句设置异常
        if n<0: raise ScoreError(str(n))
        total=total+n
    return total
#主程序并两次引用自定义异常类
#第一次初始化分数列表, 其中内含一个负数
score=[80,-86,68,92,88]
print("有异常: ")
print("平均分数: ",total(score))
#第二次初始化分数列表, 全部正数
```

```
score=[80,86,68,92,88]
print("没有异常：")
print("平均分数：",total(score)//5)
```

本例中的主程序第一次调用自定义异常时提供负数－86，系统将抛出 NumberError 异常（由第 15 行的 raise 语句抛出），程序在处理异常（显示错误信息）后将终止执行。

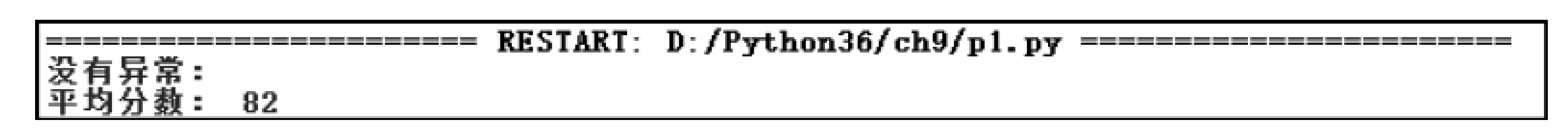
运行结果如图 9-21 所示。

```
======================= RESTART: D:/Python36/ch9/p1.py =======================
有异常：
Traceback (most recent call last):
  File "D:/Python36/ch9/p1.py", line 22, in <module>
    print("平均分数：",total(score))
  File "D:/Python36/ch9/p1.py", line 15, in total
    if n<0: raise ScoreError(str(n))
ScoreError: -86:分数不能为负数
```

图 9-21　例 9-14 的异常运行结果

若注解掉（或删除）本例中的 20～22 行，则主程序第 2 次调用自定义异常时将不会抛出异常，从而得到正确结果。

运行结果如图 9-22 所示。



```
======================= RESTART: D:/Python36/ch9/p1.py =======================
没有异常：
平均分数： 82
```

图 9-22　例 9-14 的运行结果

注意：在计算机中，功能描述通常由函数（function，也可译为功能）定义实现，而 Python 中的所有函数均是由 def 语句定义的。本书沿用计算机行业习惯，将类和对象中的函数改称为方法（method），所以在 ScoreError 类中定义的是两个方法__init__()和__str__()，而不属于 ScoreError 类的 total()则是函数。

习　题　9

一、简答题

1. 什么是异常？什么是标准异常？
2. 简述程序错误的类型。
3. 简述程序运行错误的处理方法。
4. 简述 Python 解释器抛出异常的方法。
5. 简述 Python 解释器捕捉异常的方法。
6. 什么是断言？如何使用断言处理异常。
7. 如何自定义异常类？

二、编程题

1. 设定一元二次方程的系数为 a、b、c，在计算实根前测试根的判别式并使用 raise 语句抛出异常。

2. 输入数据 a 和 b，设置断言条件是 $a \neq b$，提示信息是“这不是正方形”，若是正方形则计算面积。

3. 输入三角形的 3 条边 a、b、c，设置断言条件是 $a \neq b$、$b \neq c$、$a \neq c$，提示信息是“这不是等边三角形”，若是等边三角形则由公式计算面积。(提示：等边三角形的面积公式为 $a^2 \cdot \sin 60°/2$。)

4. 自定义异常类来处理异常，若三角形的三条边长数据不能构成三角形则抛出异常，否则计算面积。(提示：已知三角形的三条边长，求面积可以使用海伦公式。)

第 10 章　图形界面设计

图形界面(Graphical User Interface,GUI)是以图形方式呈现的计算机操作界面,与过去计算机所用的命令行界面相比,图形界面对于用户具有更好的视觉感知。从使用角度来看,图形界面能够优化软件的性能,人性化用户的操作过程,减轻用户的认知困难,提高使用效率。这是计算机能够从科学研究领域普及到公众消费层面的主要原因。

本章首先介绍 Python 图形开发库,接着介绍布局管理和各种图形界面对象(例如按钮、输入框、框架、标签、列表框、菜单、滚动条、文本框、滑动杆、面板、对话框、消息框等),最后介绍事件和事件处理程序。

10.1　Python 图形界面设计

Python 的图形界面设计是以面向对象编程为基础的,本节简单介绍 Python 图形开发库中的 Tkinter 模块,后续章节将进行详细描述。

10.1.1　Python 图形开发库

为方便程序员设计图形界面,Python 语言提供若干图形界面开发库,常用的有 Tkinter、wxPython、Jython 等,下面分别进行说明。

1. Tkinter

Tkinter 模块(又称 Tk 接口)就是 Python 系统中的标准 Tk 图形界面工具包的接口,它可以在 Windows、UNIX、Macintosh 等平台中进行调用。Tk 8.0 及其后续版本均可以实现本地窗口风格,并能够运行在绝大多数操作系统平台中。

在安装 Python 时,系统将隐含安装 Tkinter 模块。若选择自定义安装,则要仔细检查安装过程中的选项,注意不要放弃安装 Tkinter 模块。

2. wxPython

wxPython 是基于 Python 系统的一个 GUI 图形库,允许程序员自由创建完整的、功能全面的图形界面。wxPython 也是一款开源软件,并且具有完全的跨平台能力,能够支持在 Windows、UNIX、Macintosh 等平台中的调用。

3. Jython

Jython 是一种完整的计算机语言,也是 Python 语言在 Java 语言中的具体实现。Jython 程序可以实现与 Java 语言的无缝连接,即 Java 中的许多标准模块都可以由 Jython 直接使用。例如,在 Jython 设计的用户界面中就可以使用 Swing、AWT、SWT 等。另外,Jython 程序也可以编译成 Java 字节代码,用于实现在不同语言程序之间的交叉调用。

本书只介绍 Python 系统中内置的 Tkinter 模块。

10.1.2 Tkinter 的常用组件与标准属性

1. Tkinter 的常用组件

Tkinter 模块提供各种组件(或称控件),例如按钮、标签、文本框、菜单、消息框等,使用户在编写 GUI 程序中进行调用。目前,Tkinter 组件非常丰富,如表 10-1 所示。

表 10-1 Tkinter 组件

组　件	名　称	描　述
Button	按钮	表示操作
Canvas	画布	显示图形元素如线条、文本和图像
Checkbutton	多选按钮	提供多选按钮
Entry	输入框	输入单行文本
Frame	框架	显示一个矩形区域,用于表示容器
Label	标签	显示文本和位图
Listbox	列表框	显示一个字符串列表
Menubutton	菜单按钮	显示菜单项
Menu	菜单	显示菜单栏、下拉菜单和弹出菜单
Message	消息框	显示多行文本,与 Label 类似
Radiobutton	单选按钮	提供单选按钮
Scale	范围	显示数值刻度,指定输出限定范围的数字区间
Scrollbar	滚动条	当内容超过可视化区域时使用
Text	文本框	输入多行文本
Toplevel	顶级容器	提供容器,与 Frame 类似
Spinbox	滑动杆	指定输入范围,与 Entry 类似
PanedWindow	面板窗口	用于窗口布局管理的插件,内含一个或多个子组件
LabelFrame	标签框架	容器组件,用于表示窗口布局
MessageBox	消息框	显示信息的消息框

2. Tkinter 的标准属性

Tkinter 的标准属性也就是所有组件的共同属性,例如<Dimension>表示组件大小,<Color>表示组件颜色,<Font>表示组件字体,<Anchor>表示超链接,<Relief>表示组件样式,<Bitmap>表示位图文件,以及<Cursor>表示光标信息等。

10.1.3 创建窗口

1. 创建窗口

利用 Tkinter 的组件就可以创建新的窗口。在创建时,首先导入 Tkinter 模块,然后再进行 Windows 窗口操作。一般语法格式如下:

```
import tkinter
<窗口名>=tkinter.Tk()
```

这里的＜窗口名＞就是所创建窗口的名称，其后就可以加以引用。

2. 设置窗口大小

在创建好窗口后，就可以使用 geometry()函数指定窗口大小。一般语法格式如下：

```
<窗口对象>. geometry(<宽度 x 高度>)
```

说明：

(1) ＜窗口对象＞就是一个＜窗口名＞，以便后续语句加以引用；

(2) ＜宽度 x 高度＞中使用的是小写字母 x，用于指定窗口大小(以像素为单位)。

还可以使用如下两个函数来指定窗口的最大尺寸和最小尺寸。一般语法格式如下：

```
<窗口对象>. maxsize(<最大宽度>, <最大高度>)
<窗口对象>. minsize(<最小宽度>, <最小高度>)
```

这两个函数将控制窗口的大小变化区域。

【例 10-1】 创建 Windows 窗口并指定窗口大小。

源程序如下：

```
#导入 Tkinter 模块
import tkinter
win=tkinter.Tk()                        #创建名为 win 的 Windows 窗口
win.title("创建 Windows 窗口")           #设置窗口标题
win.geometry("400x160")                 #设置窗口大小
```

运行结果如图 10-1 所示。

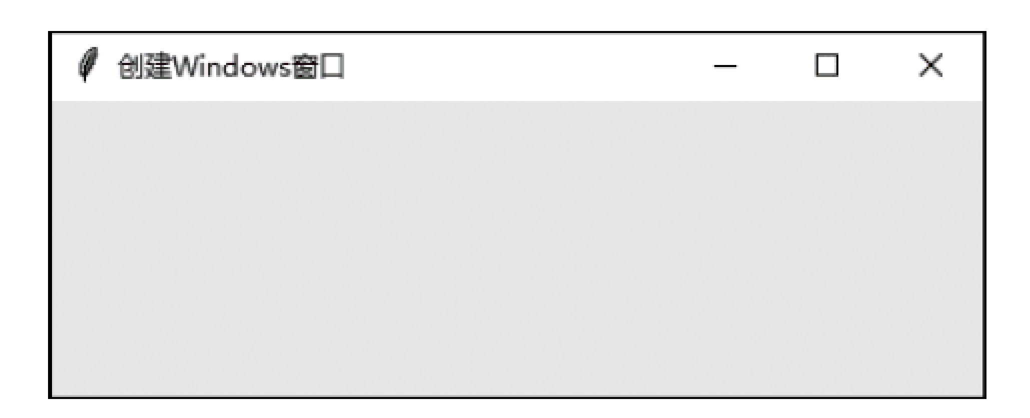


图 10-1　例 10-1 的运行结果

10.2　布局管理

布局是指一个容器中若干组件的位置安排。Tkinter 模块的布局管理分 3 种：pack、grid 和 place，因此能够合理地安排所有组件在区域中的排列和应用，实现图形界面设计。

Tkinter 不但提供了 3 种布局管理器用于组织和管理窗口内部组件，而且提供了与之对应的 3 种布局管理类：pack 类、grid 类和 place 类。

10.2.1　pack 布局的管理

在使用 pack 布局时，可向一个容器(区域)中添加组件，第一个添加的组件在最上方，然

后依次向下添加。其一般调用格式如下：

```
widget.pack(<options>)
```

这里的 widget 表示包含 pack 布局的上层窗口，<options>表示 pack 中的选项。pack 类提供许多属性和函数，由相关的组件调用，如表 10-2 和表 10-3 所示。

表 10-2　pack 类的属性

属　　性	描　　述	取　　值
fill	设置组件是否向水平或垂直方向填充	x、y、both 、none
expand	设置组件是否展开。值为 yes 时则 side 选项无效，组件显示在父容器中心位置；若 fill 选项为 both，则填充父组件的剩余空间	yes、no 或 1、0
side	设置组件的对齐方式	left、top、right、bottom
ipadx、ipady	设置 x 或 y 方向的内部间隙，即与子组件间的间隔	可设置为数值
padx、pady	设置 x 或 y 方向的外部间隙，即与并列组件间的间隔	可设置为数值
anchor	当可用空间大于所需尺寸时决定组件被放置的容器位置	n、e、s、w、nw、ne、sw、se、center

表 10-3　pack 类的函数

函　　数	描　　述
pack_slaves()	以列表方式返回本组件的所有子组件对象
pack_configure(<option>=value)	给 pack 布局管理器设置属性
propagate(boolean)	设置 True 表示父组件的几何大小由子组件决定，反之无关
pack_info()	返回 pack 提供选项所对应的值
pack_forget()	隐藏组件并忽略原设置，保留对象
location(x,y)	测试坐标点(x,y)是否在单元格中，(−1,−1)表示不在
size()	返回组件所包含的单元格，表示组件大小

【例 10-2】 pack 布局的管理示例。

源程序如下：

```
import tkinter
root=tkinter.Tk()
#设置窗口标题
root.title("pack布局管理窗口")
#设置 Label 组件
label=tkinter.Label(root,text="pack布局管理窗口\n")
#将 Label 组件添加至窗口
label.pack()
#创建 Button 组件按钮_1，并将组件 btn_1 添加至窗口且顶端对齐
btn1=tkinter.Button(root,text="按钮_1")
```

```
btn1.pack(side=tkinter.TOP)
#创建 Button 组件按钮_2，并将组件 btn_2 添加至窗口且左对齐
btn2=tkinter.Button(root,text="按钮_2")
btn2.pack(side=tkinter.LEFT)
#创建 Button 组件按钮_3，并将组件 btn_3 添加至窗口且右对齐
btn3=tkinter.Button(root,text="按钮_3")
btn3.pack(side=tkinter.RIGHT)
#创建 Button 组件按钮_4，并将组件 btn_4 添加至窗口且底端对齐
btn4=tkinter.Button(root,text="按钮_4")
btn4.pack(side=tkinter.BOTTOM)
root.mainloop()
```

运行结果如图 10-2 所示。

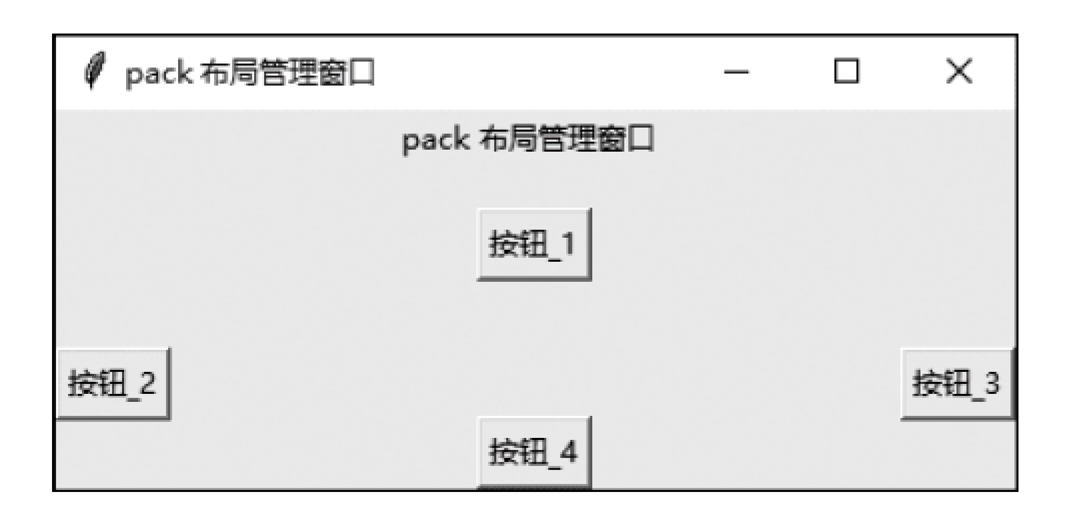


图 10-2　例 10-2 的运行结果

从图 10-2 中可以发现，使用 pack 布局时，安排组件的顺序是从左到右、从上到下的，每一个组件都可自行选择垂直对齐或水平对齐的方式。

10.2.2　grid 布局的管理

grid 布局又称为网格布局，是最常用的布局管理。由于程序通常使用矩形界面，为此可以将矩形划分为若干行列的网格，然后根据行号和列号，将若干全部组件分别放置在网格中。其一般调用格式如下：

```
widget.grid(<options>)
```

这里的 widget 是表示包含 grid 布局的窗口，<options>表示 grid 中的选项。另外，使用 grid 布局时，需要分别用参数 row 表示行，用参数 column 表示列，且 row 和 column 均是从 0 开始编号的。

grid 类提供许多属性，由相关组件调用，其中 ipadx、ipady、padx 和 pady 属性与 pack 类中的属性相同，新增的属性如表 10-4 所示。grid 类提供的函数与 pack 类中的函数相同，由相关组件调用。

表 10-4　grid 类的属性

属　性	描　述	取　值
row	设置组件所在行数	取值为整数
column	设置组件所在列数	取值为整数

续表

属　　性	描　　述	取　　值
sticky	设置组件在网格中的对齐方式	n、e、s、w、nw、ne、sw、se、center
rowspan	组件所跨越的行数	跨越行数
columnspan	组件所跨越的列数	跨越列数

【例 10-3】 grid 布局管理示例。

源程序如下：

```
from tkinter import *
root=Tk()
#设置主窗口大小为 320×160
root.geometry("320x160")
root.title("显示幻方")
#设计 grid 布局管理内含的 9 个按钮
btn_1=Button(root,text="8",width=5,bg="yellow")
btn_2=Button(root,text="1",width=5)
btn_3=Button(root,text="6",width=5)
btn_4=Button(root,text="3",width=5)
btn_5=Button(root,text="5",width=5,bg="green")
btn_6=Button(root,text="7",width=5)
btn_7=Button(root,text="4",width=5)
btn_8=Button(root,text="9",width=5)
btn_9=Button(root,text="2",width=5,bg="gray")
#将 9 个按钮按指定行列位置放置
btn_1.grid(row=0,column=0)              #按钮放置在 0 行 0 列
btn_2.grid(row=0,column=1)              #按钮放置在 0 行 1 列
btn_3.grid(row=0,column=2)              #按钮放置在 0 行 2 列
btn_4.grid(row=1,column=0)              #按钮放置在 1 行 0 列
btn_5.grid(row=1,column=1)              #按钮放置在 1 行 1 列
btn_6.grid(row=1,column=2)              #按钮放置在 1 行 2 列
btn_7.grid(row=2,column=0)              #按钮放置在 2 行 0 列
btn_8.grid(row=2,column=1)              #按钮放置在 2 行 1 列
btn_9.grid(row=2,column=2)              #按钮放置在 2 行 2 列
root.mainloop()
```

运行结果如图 10-3 所示。

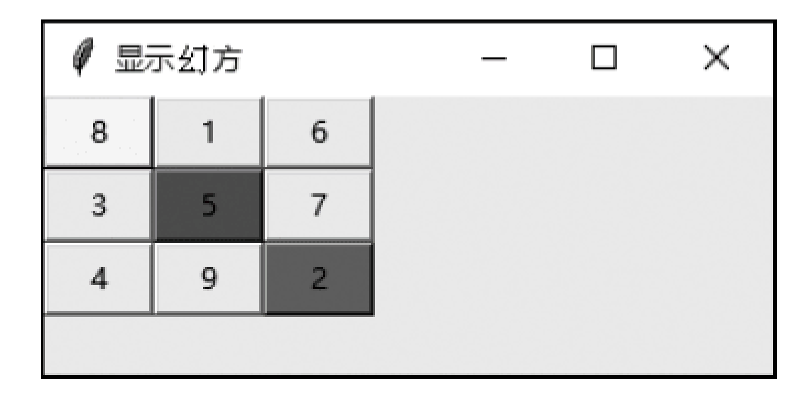


图 10-3　例 10-3 的运行结果

从图 10-3 中可以发现，grid 布局是将所有组件组织在一个矩形网格中的，每个网格由行标和列标标识。

10.2.3 place 布局的管理

使用 place 布局时，允许指定组件的大小与位置。place 的优点是可以精确控制组件的位置，不足之处是改变窗口大小时，子组件不能随之灵活改变大小。其一般调用格式如下：

```
widget.place(<options>)
```

这里的 widget 表示包含 place 布局，<options>表示 place 中的选项。

place 类中提供许多属性和函数，由相关的组件调用，其中的属性如表 10-5 所示。

表 10-5　place 类的属性

属　　性	说　　明	取　　值
anchor	锚选项，同 pack 类	默认值为 nw
x、y	组件左上角的 x、y 坐标	整数，默认为 0
relx、rely	组件相对于父容器的 x、y 坐标	0～1 的浮点数
width、height	组件的宽度和高度	非负整数
relwidth、relheight	组件相对于父容器的宽度和高度	0～1 的浮点数
bordermode	若为 inside，则组件内部大小和位置为相对且无边框；若为 outside，则组件外部大小为相对且有边框	inside、outside

place 类中的函数如表 10-6 所示。

表 10-6　place 类的函数

函　　数	描　　述
place_slaves()	以列表方式返回本组件的所有子组件
place_configure(<option>=value)	给 place 布局设置属性
propagate(boolean)	设置为 True 表示父组件的几何大小由子组件决定
place_info()	返回 pack 提供的选项所对应的值
grid_forget()	隐藏组件并忽略原设置，保留对象
location(x, y)	测试坐标点(x,y)是否在单元格中，(−1,−1)表示不在
size()	返回组件所包含的单元格，表示组件大小

【例 10-4】 place 布局的管理示例。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("注册界面")
root["width"]=200
```

```
root["height"]=80
#指定第 1 个 Label 组件的坐标为(1, 1)
Label(root,text="账号",width=8).place(x=1,y=1)
#指定第一个 Entry 组件的坐标为(45, 1)
Entry(root,width=24).place(x=45,y=1)
#指定第二个 Label 组件的坐标为(1, 20)
Label(root,text="密码",width=8).place(x=1,y=20)
#指定第二个 Entry 组件的坐标为(45, 20)
Entry(root,width=24, show="*").place(x=45,y=20)
#指定第一个 Button 组件的坐标为(48, 60)
Button(root,text="注册",width=6).place(x=48,y=60)
#指定第二个 Button 组件的坐标为(164, 60)
Button(root,text="放弃",width=6).place(x=164,y=60)
```

运行结果如图 10-4 所示。



图 10-4　例 10-4 的运行结果

从图 10-4 中可以发现，place 布局是以坐标(以像素为单位)点为准安放组件的。另外，显示密码时的星号是由 Entry 组件的 show 属性设置的。Python 并没有提供专门的密码组件。

上面介绍了 Tkinter 模块的整体布局，下面介绍 Tkinter 模块中的各个组件。

10.3　Tkinter 的常用组件

从设计图形界面角度来看，Tkinter 的常用组件可以分为两类，一类提供文本、图形与图像的显示，另一类在组件上实现人机交互操作，后者就是事件处理，本章最后将专门介绍。

在 Python 编程规范中，通常是首先导入 Tkinter 模块并创建窗口，然后定义事件处理函数和普通函数，最后设计组件及其操作模式。这样就需要反复查看程序，以便清楚程序控制的流程。

10.3.1　Label 组件

Label(标签)组件用于在窗口中显示文本或位图。常用属性如表 10-7 所示。

表 10-7　Label 组件的属性

属　　性	说　　明	属　　性	说　　明
anchor	设置标签中文本的位置	bitmap	设置标签中的位图文件
background(＜bg＞)	设置背景色	font	设置字体
foreground(＜fg＞)	设置前景色	image	设置标签中的图像文件
borderwidth(＜bd＞)	设置边框宽度	justify	多行文本的对齐方式
width	设置标签宽度	text	设置标签中的文本
height	设置标签高度	textvariable	显示文本自动更新

【例 10-5】 Label 组件的使用。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("Label 组件示例")
#第 1 次创建含文字的 Label 组件
label_1=Label(root,text="Python 程序设计",anchor="nw")
#显示第 1 个 Label 组件
label_1.pack()
#第 2 次创建含文字的 Label 组件
label_2=Label(root,text="计算思维视角",anchor="nw")
#显示第 2 个 Label 组件
label_2.pack()
root.mainloop()
```

运行结果如图 10-5 所示。



图 10-5 例 10-5 的运行结果

从图 10-5 中可以发现，Label 组件的排列顺序是从上到下的。

10.3.2 Button 组件

Button（按钮）组件是最常用的 Tkinter 部件之一，用于表示各种按钮。按钮可以是文本也可以是图像，使用 command 属性可将要调用的函数绑定到按钮，以实现事件处理。当单击按钮事件发生时，系统将自动调用指定函数来实现事件处理，参见 10.4 节。

Button 组件的属性如表 10-8 所示。

表 10-8 Button 组件的属性

属　性	说　明
anchor	设置按钮上文本的位置
background(＜bg＞)	设置按钮的背景色
bitmap	设置按钮上显示的位图
borderwidth(＜bd＞)	设置按钮边框的宽度
command	设置按钮消息的调用函数
cursor	设置鼠标移动到按钮上的指针样式
font	设置按钮上文本的字体
foreground(＜fg＞)	设置按钮的前景色

续表

属　　性	说　　明
height	设置按钮的高度
image	设置按钮上显示的图片
state	设置按钮的状态
text	设置按钮上显示的文本
width	设置按钮的宽度
padx	设置文本与按钮边框 x 的距离
pady	设置文本与按钮边框 y 的距离
activeforeground	按下时的前景色
textvariable	可变文本

【例 10-6】 Button 组件示例。

创建一个含有 4 个 Button 组件的界面，并设置 width、height、relief、bg、bd、fg、state、bitmap、anchor 等属性。

源程序如下：

```
from tkinter import *
#导入消息框组件
from tkinter.messagebox import *
root=Tk()
root.title("Button 组件示例")
#创建第 1 个 Button 组件并设置属性
Button(root,text="按钮 1",width=40,relief=GROOVE,bg="white").pack()
#创建第 2 个 Button 组件并设置属性
Button(root,text="按钮 2",width=40,state=DISABLED).pack()
#创建第 3 个 Button 组件并设置属性
Button(root,text="按钮 3",fg="blue",width=40).pack()
#创建第 4 个 Button 组件并设置属性
Button(root,text="按钮 4",anchor='sw',width=40).pack()
root.mainloop()
```

运行结果如图 10-6 所示。

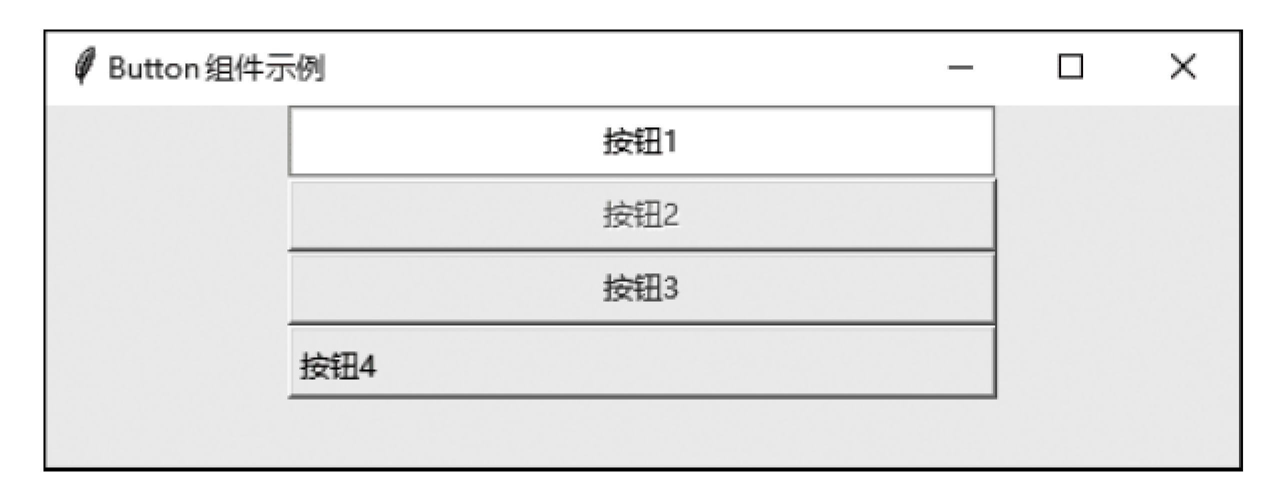


图 10-6　例 10-6 的运行结果

从图 10-6 中可以发现，Button 组件的排列顺序是从左到右、从上到下的。

10.3.3　Entry 和 Text 组件

下面分别介绍 Entry(单行文本框)组件和 Text(多行文本框)组件。

1. Entry 和 Text 组件的属性

Entry 和 Text 组件的属性如表 10-9 所示。

表 10-9　Entry 和 Text 组件的属性

属　性	说　明
background(<bg>)	设置文本框背景色
foreground(<fg>)	设置文本框前景色
selectbackground	选定文本背景色
selectforeground	选定文本前景色
borderwidth(<bd>)	设置文本框的边框宽度(以像素为单位)
font	设置字体
show	文本框显示的字符，若为星号则表示为密码框
state	设置状态
width、height	设置文本框的宽度和高度
textvariable	可变文本

2. Entry 组件

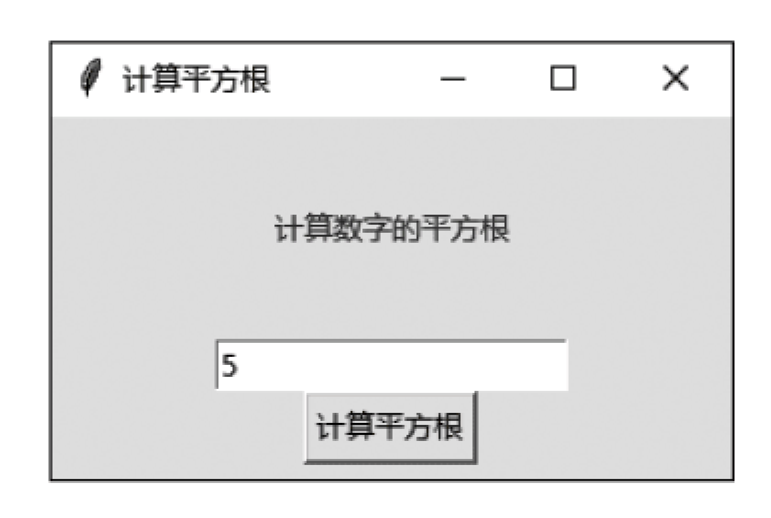
【例 10-7】 Entry 组件示例。

源程序如下：

```
import tkinter, math
#定义事件处理函数
def click_btn():
    #获取文本框内输入的内容转换成浮点数
    n=float(data.get())
    label_temp.config(text="%.2f 的平方根=%.2f" % (n,math.sqrt()))
root=tkinter.Tk()
root.title("计算平方根")
#创建 Label 组件
label_temp=tk.Label(root,text="计算数字的平方根",height=5,width=20,fg="blue")
label_temp.pack()
#创建 Entry 组件
data=tk.Entry(root)
data.pack()
#按钮操作及其事件处理
tran=tk.Button(root,text="计算平方根",command=click_btn)
tran.pack()
```

```
root.mainloop()
```

运行结果如图 10-7 所示。



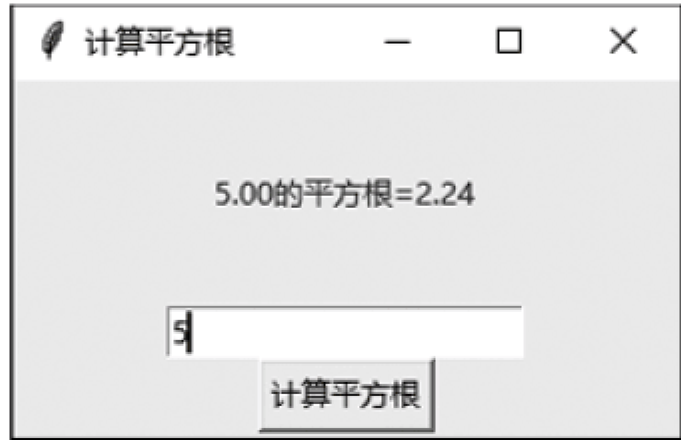


图 10-7　例 10-7 的运行结果

在图 10-7(a)所示的 Entry 组件中输入整数 5 并单击"计算平方根"Button 组件后，系统将由 Label 组件显示计算结果，如图 10-7(b)所示。在 Entry 组件中输入 5 不是事件，而单击"计算平方根"Button 组件将产生单击事件，并执行 click_btn()函数来实现事件处理。要注意的是，这里的函数调用是由 command 属性实现的，且函数没有用圆括号界定。

3. Text 组件的使用

Text 组件是通过设置行列数量来表示的。

【例 10-8】　Text 组件示例。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("Text 组件示例")
#设置 40 列×7 行的多行文本框
tx=Text(root,width=40,height=7)
tx.pack()
#由 INSERT 属性表示在光标位置插入字符串：Python 程序设计
tx.insert(INSERT,"Python 程序设计\n")
#由 END 属性表示在末尾处插入两个字符串
tx.insert(END,"计算思维视角\n")
tx.insert(END,"清华大学出版社\n")
def output():
    print("这是一本教材")
#创建 Button 组件并插入到组件 Text 中
btn=Button(tx,text="单击",command=output)
tx.window_create(INSERT,window=btn)
mainloop()
```

运行结果如图 10-8 所示。

单击图 10-8 中的"单击"按钮后，系统将在 IDLE 交互环境中显示"这是一本教材"。

思考：在多行文本框中自行输入若干行字符串后，show()函数应该如何获取字符串并显示，请修改程序。

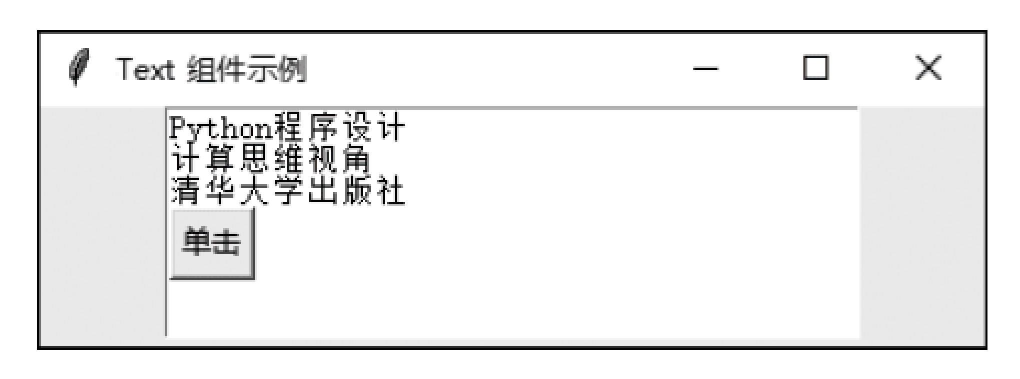


图 10-8　例 10-8 的运行结果

10.3.4　Listbox 组件

Listbox(列表框)组件用于显示多个条目,并且允许用户选择一个或多个条目。

【例 10-9】　Listbox 组件示例(1)。

以下程序将获取列表框组件 Listbox 中的条目。

源程序如下:

```
from tkinter import *
root=Tk()
#将字符串与 Listbox 组件的值绑定
item=StringVar()
#获取列表框中的全部条目
def click_1():
    print(item.get())
#获取列表框中的所选条目
def click_2():
    for i in lb.curselection():
        print("显示一个条目: ",lb.get(i))
root.title("Listbox 组件示例")
lb=Listbox(root,listvariable=item)
for i in ["Java","JavaScript","Python"]:
    lb.insert(END,i)
lb.pack()
#创建 Button 组件
btn_1=Button(root,text="获取列表框中的全部条目",command=click_1,width=24)
btn_1.pack()
#创建 Button 组件
btn_2=Button (root,text="获取列表框中的所选条目",command=click_2,width=24)
btn_2.pack()
root.mainloop()
```

运行结果如图 10-9 所示。

如图 10-9 所示的 3 行信息只能由程序设置,这里由列表实现。在选择列表框中的 Python 条目后,再单击列表框中的"获取列表框中的所选条目"按钮,则将得到显示结果"Python";在单击列表框中的"获取列表框中的全部条目"按钮后,运行结果如图 10-10 所示。

【例 10-10】　Listbox 组件示例(2)。

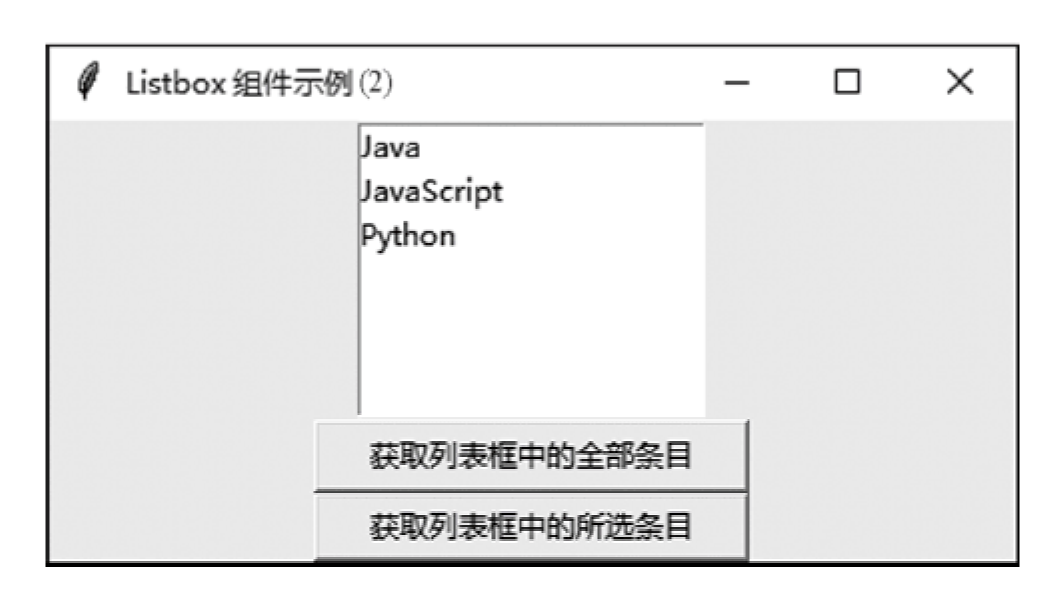


图 10-9 例 10-9 的初始界面

```
====================== RESTART: D:\Python36\cha\p1.py ======================
显示一个条目： Python
('Java', 'JavaScript', 'Python')
```

图 10-10 例 10-9 的运行结果

下面是将一个列表框中的条目添加到另一个列表框中，以及删除列表框中的条目。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("Listbox 组件示例(2)")
#定义"添加"按钮的事件处理函数
def click_1():
    for i in lbox_1.curselection():        #遍历选中项
lbox_2.insert(0,lbox_1.get(i))             #添加到右侧列表框
#定义"删除"按钮的事件处理函数
def click_2():
    for i in lbox_2.curselection():        #遍历选中项
lbox_2.delete(i)                           #从右侧列表框中删除
#初始化列表以表示列表框中的条目
lst=["Java","C","C++","C#","Python","JavaScript"]
#创建两个列表框组件
lbox_1=Listbox(root)
lbox_2=Listbox(root)
#将全部列表元素插入到左侧列表框中
for i in lst:
lbox_1.insert(0,i)
#将列表框组件放置到窗口中
lbox_1.grid(row=0,column=0,rowspan=2)
#创建两个 Button 组件
btn_1=Button(root,text="添加>>",command=click_1,width=20)
btn_2=Button(root,text="删除<<",command=click_2,width=20)
#显示两个 Button 组件
btn_1.grid(row=0,column=1,rowspan=2)
btn_2.grid(row=1,column=1,rowspan=2)
lbox_2.grid(row=0,column=2,rowspan=2)
```

```
root.mainloop()
```

运行结果如图 10-11 所示。

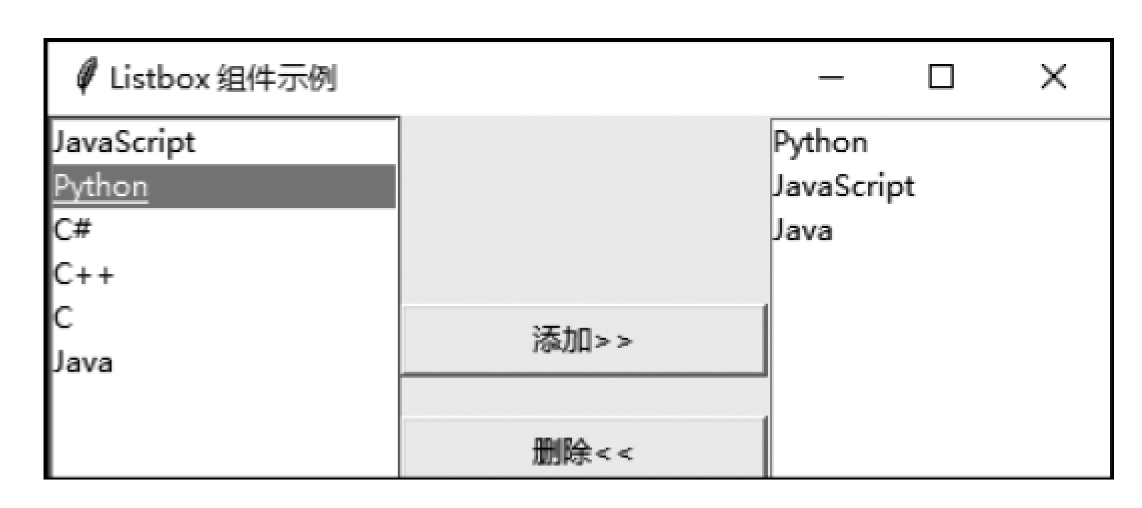


图 10-11　例 10-10 的运行结果

在图 10-11 中，由程序初始设定 6 个计算机语言并显示在左侧列表框中。若在选定条目后单击“添加”按钮，则该条目将出现在右侧列表框中。若在右侧列表框选定条目后单击“删除”按钮，则该条目将从右侧列表框中被删除，即左侧列表框中的条目是固定的。

10.3.5　Radiobutton 和 Checkbutton 组件

Radiobutton(单选按钮)和 Checkbutton(复选按钮)组件分别实现在若干选项中的单选操作和复选操作。Radiobutton 组件用于同一组单选按钮中选择一个单选按钮(不能同时选定多个)，Radiobutton 组件可以显示文本，也可以显示图像。Checkbutton 组件用于选择一项或多项，同样 Checkbutton 组件可以显示文本，也可以显示图像。

1. Radiobutton 和 Checkbutton 组件的属性

Radiobutton 和 Checkbutton 组件的属性如表 10-10 所示。

表 10-10　Radiobutton 和 Checkbutton 组件的属性

属　　性	说　　明
anchor	设置文本位置
background(＜bg＞)	设置背景色
foreground(＜fg＞)	设置前景色
borderwidth	设置边框宽度
width	设置组件宽度
height	设置组件高度
bitmap	设置组件中的位图文件
image	设置组件中的图像文件
font	设置字体
justify	设置组件中多行文本的对齐方式
text	设置组件的文本
value	设置组件被选中中关联变量的值
variable	设置组件所关联的变量

续表

属　　性	说　　明
indicatoron	控制参数，为 0 时组件会被绘制成按钮形式
textvariable	可变文本显示

2. Radiobutton 组件

【例 10-11】 Radiobutton 组件示例。

源程序如下：

```
import tkinter
root=tkinter.Tk()
root.title("Radiobutton 组件示例")
#创建 StringVar 对象
rt=tkinter.StringVar()
#设置初始值为 1，初始选中的是"四川成都"
rt.set("1")
rad=tkinter.Radiobutton(root,variable=rt,value="1",text="四川成都")
rad.pack()
rad=tkinter.Radiobutton(root,variable=rt,value="2",text="山西太原")
rad.pack()
rad=tkinter.Radiobutton(root,variable=rt,value="3",text="江苏南京")
rad.pack()
rad=tkinter.Radiobutton(root,variable=rt,value="4",text="辽宁沈阳")
rad.pack()
rad=tkinter.Radiobutton(root,variable=rt,value="5",text="陕西西安")
rad.pack()
rad=tkinter.Radiobutton(root,variable=rt,value="6",text="西藏拉萨")
rad.pack()
rad=tkinter.Radiobutton(root,variable=rt,value="7",text="广西南宁")
rad.pack()
root.mainloop()
```

运行结果如图 10-12 所示。



图 10-12　例 10-11 的运行结果

3. Checkbutton 组件

【例 10-12】 Radio 与 Checkbutton 组件示例。

源程序如下：

```
import tkinter as tk
rt=tk.Tk()
rt.title("Radiobutton 与 Checkbutton 组件示例")
#获取单选的颜色
def colors():
    label_1.config(fg=c.get())
#获取复选的字体
def types():
textType=typeBlod.get()+typeItalic.get()
    if textType==1:
        label_1.config(font=("Arial",12,"bold"))
    if textType==2:
        label_1.config(font=("Arial",12,"italic"))
label_1=tk.Label(rt,text="查看样式: check the style",height=3,font=("Arial",12))
#设置初始颜色为蓝色
label_1.config(fg="blue")
label_1.pack()
#设置 Radiobutton 按钮的颜色变量 color
c=tk.StringVar()
c.set("blue")
tk.Radiobutton(rt,text="红",variable=c,value="red",command=colors).pack(side=
tk.LEFT)
tk.Radiobutton(rt,text="蓝",variable=c,value="blue",command=colors).pack(side
=tk.LEFT)
tk.Radiobutton(rt,text="绿",variable=c,value="green",command=colors).pack(side
=tk.LEFT)
#定义 typeBlod 变量表示粗体文字
typeBlod=tk.IntVar()
#定义 typeItalic 变量表示斜体文字
typeItalic=tk.IntVar()
tk.Checkbutton(rt,text="粗体",variable=typeBlod,command=types).pack(side=tk.
LEFT)
tk.Checkbutton(rt,text="斜体",variable=typeItalic,command=types).pack(side=tk.
LEFT)
rt.mainloop()
```

选中“红色”按钮和“粗体”按钮后，运行结果如图 10-13 所示。

10.3.6 Frame 与 LabelFrame 组件

在需要分组(区域)设计时，Frame(框架)与 LabelFrame(标签框架)组件，负责安排其他组件的位置。Frame 与 LabelFrame 组件在图形界面上显示为一个矩形区域，用于安排相关



图 10-13　例 10-12 的运行结果

组件。

【例 10-13】　Frame 与 LabelFrame 组件示例。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("Frame 与 LabelFrame 组件示例")
#创建框架组件并显示
frm=Frame(root)
frm.pack()
#创建第 1 个 LabelFrame 组件并显示
lbfrm_1=LabelFrame(root,text="第 1 个标签框架组件")
lbfrm_1.pack()
#创建第 2 个 Frame 组件并在窗口底部显示
lbfrm_2=LabelFrame(root,text="第 2 个标签框架组件")
lbfrm_2.pack(side=BOTTOM )
#在第 1 个 Frame 组件中设置 3 个按钮，颜色分别为蓝、绿、蓝
btn_1=Button(frm,text="蓝色",fg="blue")
btn_1.pack( side=LEFT)
btn_2=Button(frm,text="绿色",fg="green")
btn_2.pack( side=LEFT )
btn_3=Button(frm,text="蓝色",fg="blue")
btn_3.pack( side=LEFT )
#在第 2 个 Frame 组件中设置了个按钮，颜色为红
btn_3=Button(lbfrm_1,text="红色",fg="red")
btn_3.pack()
#在第 3 个 Frame 组件中设置一个按钮，颜色为黑
btn_4=Button(lbfrm_2,text="黑色",fg="black")
btn_4.pack()
root.mainloop()
```

运行结果如图 10-14 所示。

10.3.7　Scrollbar 组件

Scrollbar(滚动条)组件用于在一个可见范围内滚动一些条目，根据方向可分为垂直滚动条和水平滚动条。Scrollbar 组件常常被用于滚动字符串、画布与列表框中的条目等。

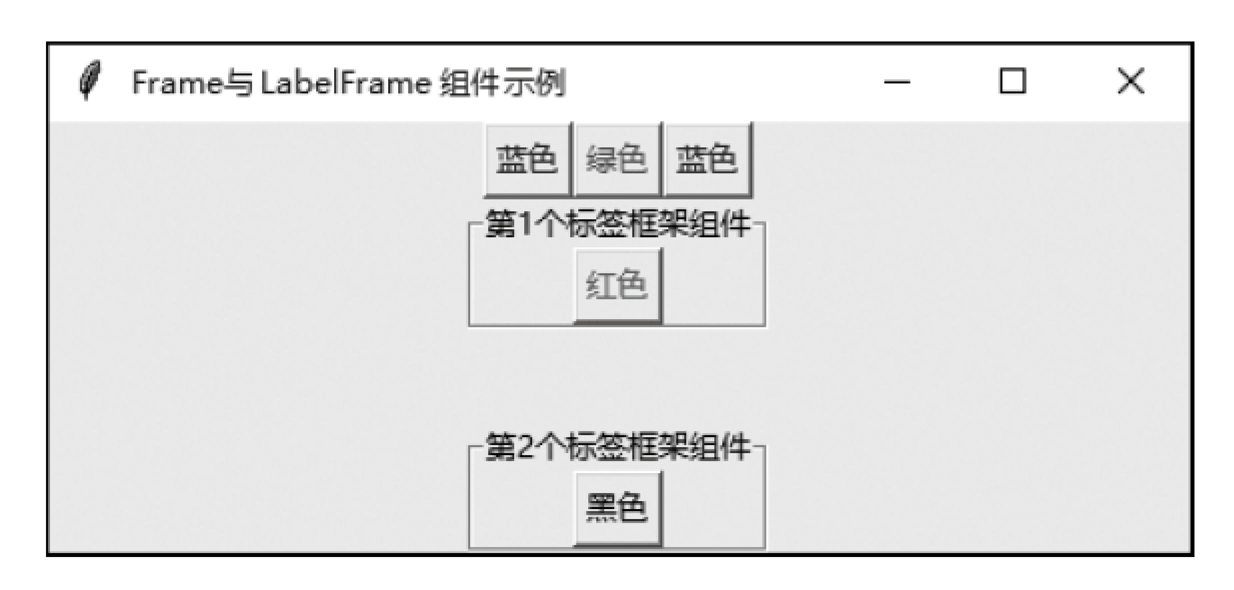


图 10-14　例 10-13 的运行结果

Scrollbar 组件通常与 Text、Canvas 和 Listbox 组件一起使用，水平滚动条还能跟 Entry 组件配合使用。

【例 10-14】 Scrollbar 组件示例。

源程序如下：

```
from tkinter import *
#定义事件处理函数，鼠标松开事件显示当前选中的条目
def disp_item(event):
    print (lst.get(lst.curselection()))
root=Tk()
root.title("scrlbar 组件示例")
#创建列表框
lst=Listbox(root)
lst.bind("<ButtonRelease-1>",disp_item)
#列表框内追加 60 项条目
for i in range(1,61,1):
    lst.insert(END,"   行号："+str(i))
#设置垂直滚动条
lst.pack(side=LEFT,fill=BOTH)
scrlbar=scrollbar(root)
scrlbar.pack(side=RIGHT,fill=Y)
scrlbar.config(command=lst.yview)
lst.configure(yscrollcommand=scrollbar.set)
root.mainloop()
```

运行结果如图 10-15 所示。

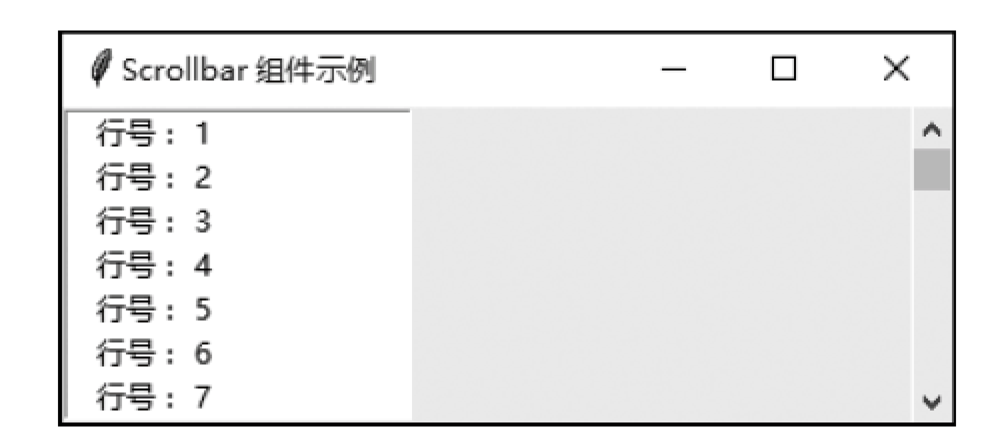


图 10-15　例 10-14 的运行结果 1

在图 10-15 中，可以调整窗口和垂直滚动条得到前 7 个条目，若分别单击其中的 3 条，

则可 IDLE 交互环境中获得如图 10-16 所示的运行结果。

```
======================= RESTART: D:\Python36\cha\p1.py =======================
    行号: 5
    行号: 7
    行号: 2
```

图 10-16　例 10-14 的运行结果 2

10.3.8　Menu 组件

图形界面程序通常提供菜单,菜单包含各种按照主题分组的基本命令。图形界面程序包括两种菜单。

(1) 主菜单。提供窗体的菜单系统。通过单击可以下拉出子菜单,选中选项后可执行相关的操作。常用的主菜单通常包括文件、编辑、视图、帮助等选项。

(2) 下拉菜单。选中主菜单上的选项,弹出的子菜单就是下拉菜单,其中列出了与该对象相关的常用菜单命令,例如剪切、复制、粘贴等。

要实现菜单功能就要使用 Menu 组件。

1. Menu 组件的属性和函数

Menu(菜单)组件的属性和函数如表 10-11 和表 10-12 所示。

表 10-11　组件 Menu 的属性

属　　性	说　　明
tearoff	分隔窗口,为 0 表示原窗口,为 1 表示点击为两个窗口
bg	设置背景色
fg	设置前景色
borderwidth	设置边框宽度
font	设置字体
activebackground	设置点击时的背景色
activeforeground	设置点击时的前景色
cursor	设置光标位置
postcommand	设置命令处理
selectcolor	设置选中时的背景色
title	设置标题
type	设置类型

表 10-12　Menu 组件的 Menu()函数及其参数

函　　数	说　　明
add_cascade	添加子选项
add_command	添加命令(label 参数为显示内容)
add_separator	添加分隔线

续表

函　　数	说　　明
add_checkbutton	添加确认按钮
delete	删除子选项

2. 创建主菜单

【例 10-15】 用 Menu 组件创建主菜单。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("Menu 组件示例")
#菜单项事件函数，可以为每个菜单项单独编程
#定义事件处理函数
def info():
    print("主菜单已经单击")
mnm=Menu(root)
for i in ["文件","编辑","视图","选项","工具","窗口","帮助"]:
    #添加菜单项
    mnm.add_command(label=i,command=info)
#添加到主菜单窗口中
root["menu"]=mnm
root.mainloop()
```

运行结果如图 10-17 所示。

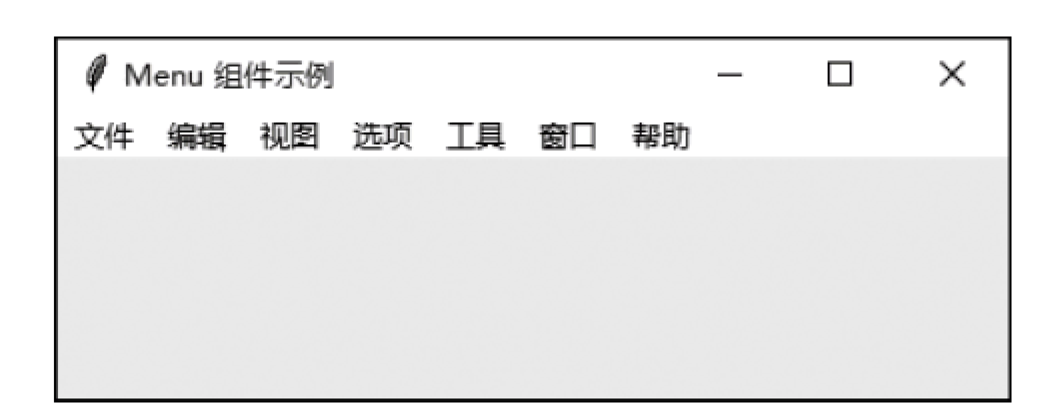


图 10-17　例 10-15 的运行结果

单击图 10-17 中的任何一个菜单后，系统将显示：主菜单已经单击。

若要在主菜单中设计下拉菜单，则应该使用下面介绍的 Menu()函数。

3. 创建下拉菜单

【例 10-16】 用 Menu 组件创建下拉菜单。

源程序如下：

```
from tkinter import *
#定义事件处理函数
def info():
    print("这是下拉菜单示例")
root=Tk()
```

```
root.title("下拉菜单示例")
mnu=Menu(root)                        #创建主菜单
#创建下拉菜单
filemenu=Menu(mnu)
editmenu=Menu(mnu)
#添加下拉菜单项
for i in ["打开","关闭","退出"]:
    filemenu.add_command(label=i,command=info)
#添加下拉菜单项
for i in ["复制","剪切","粘贴"]:
    editmenu.add_command(label=i,command=info)
#将 filemenu 作为文件下拉菜单
mnu.add_cascade(label="文件",menu=filemenu)
#将 editmenu 作为编辑下拉菜单
mnu.add_cascade(label="编辑",menu=editmenu)
mnu.add_cascade(label="视图",menu=editmenu)
mnu.add_cascade(label="选项",menu=editmenu)
mnu.add_cascade(label="窗口",menu=editmenu)
mnu.add_cascade(label="帮助",menu=editmenu)
#添加到主菜单窗口中
root["menu"]=mnu
root.mainloop()
```

运行结果如图 10-18 所示。

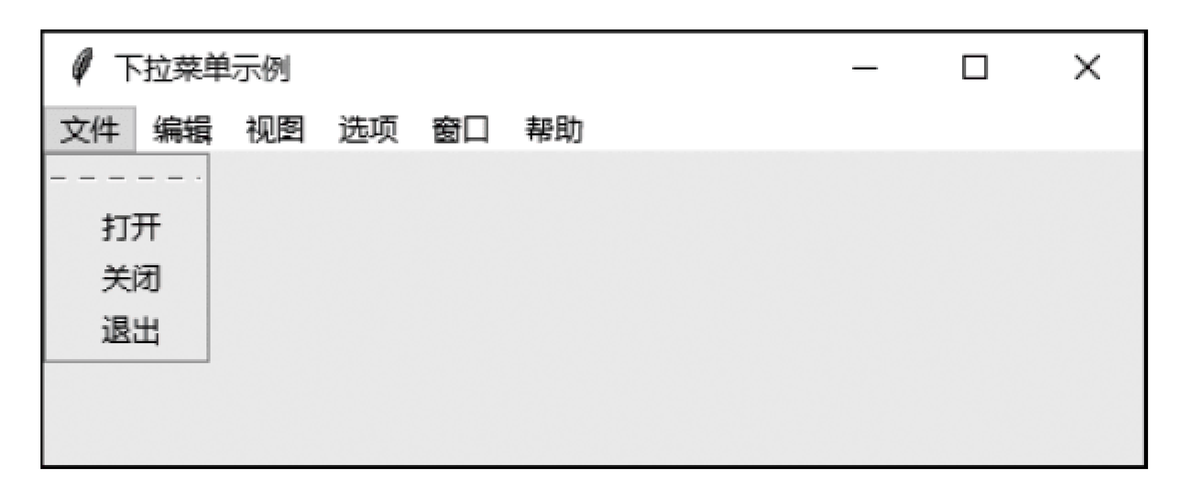


图 10-18　例 10-16 的运行结果

选中“文件”|“关闭”菜单选项，系统将显示运行结果。本例为下拉菜单示例，不是真正的复制操作，这里只是设计图形界面，而关闭文件操作在后面介绍。

为书写简洁起见，后 4 个菜单均将 editmenu 作为下拉菜单。

10.3.9　对话框

对话框用于与用户交互和检索信息。通过 messagebox、filedialog 和 simpledialog 组件是通用的预定义对话框；用户可以通过继承 TopLevel（顶层容器）类来创建自定义对话框。

1. 创建文件对话框

filedialog（文件对话框）组件的属性如表 10-13 所示。

表 10-13 filedialog 的属性

参　数	说　明	参　数	说　明
filetype	指定文件类型	initialfile	指定默认文件
initialdir	指定默认目录	title	指定文件对话框的标题

【例 10-17】 filedialog 组件示例。

源程序如下：

```
from tkinter import *
#导入 tkinter.filedialog 模块
from tkinter.filedialog import *
root=Tk()
root.title("filedialog 组件示例")
#定义"打开文件"按钮事件处理函数
def openfile():
    #显示“打开文件”对话框，返回选中文件名以及路径
    r= askopenfilename(title="打开文件",filetypes=[("Python","*.py"),("All
Files","*.*")])
    print(r)
#定义"保存文件"按钮事件处理函数
def savefile():
    #显示“保存文件”对话框
    r=asksaveasfilename(title="保存文件",initialdir="d:\Python36",initialfile=
"p1.py")
    print(r)
#设置窗口大小
root.geometry("400x200")
#创建两个 Button 组件
btn_1=Button(root,text="打开文件",command=openfile)
btn_2=Button(root,text="保存文件",command=savefile)
btn_1.pack(side="left")
btn_2.pack(side="right")
root.mainloop()
```

运行结果如图 10-19 所示。单击“打开文件”按钮后，系统将显示“打开文件”对话框，如图 10-20 所示。

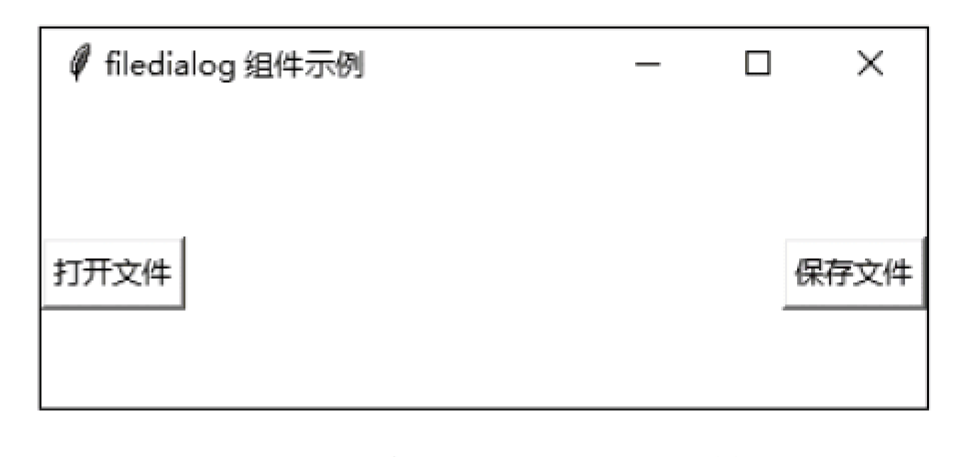


图 10-19　例 10-17 的运行结果



图 10-20　打开文件对话框

单击“保存文件”按钮后，系统将显示保存文件对话框，如图 10-21 所示。

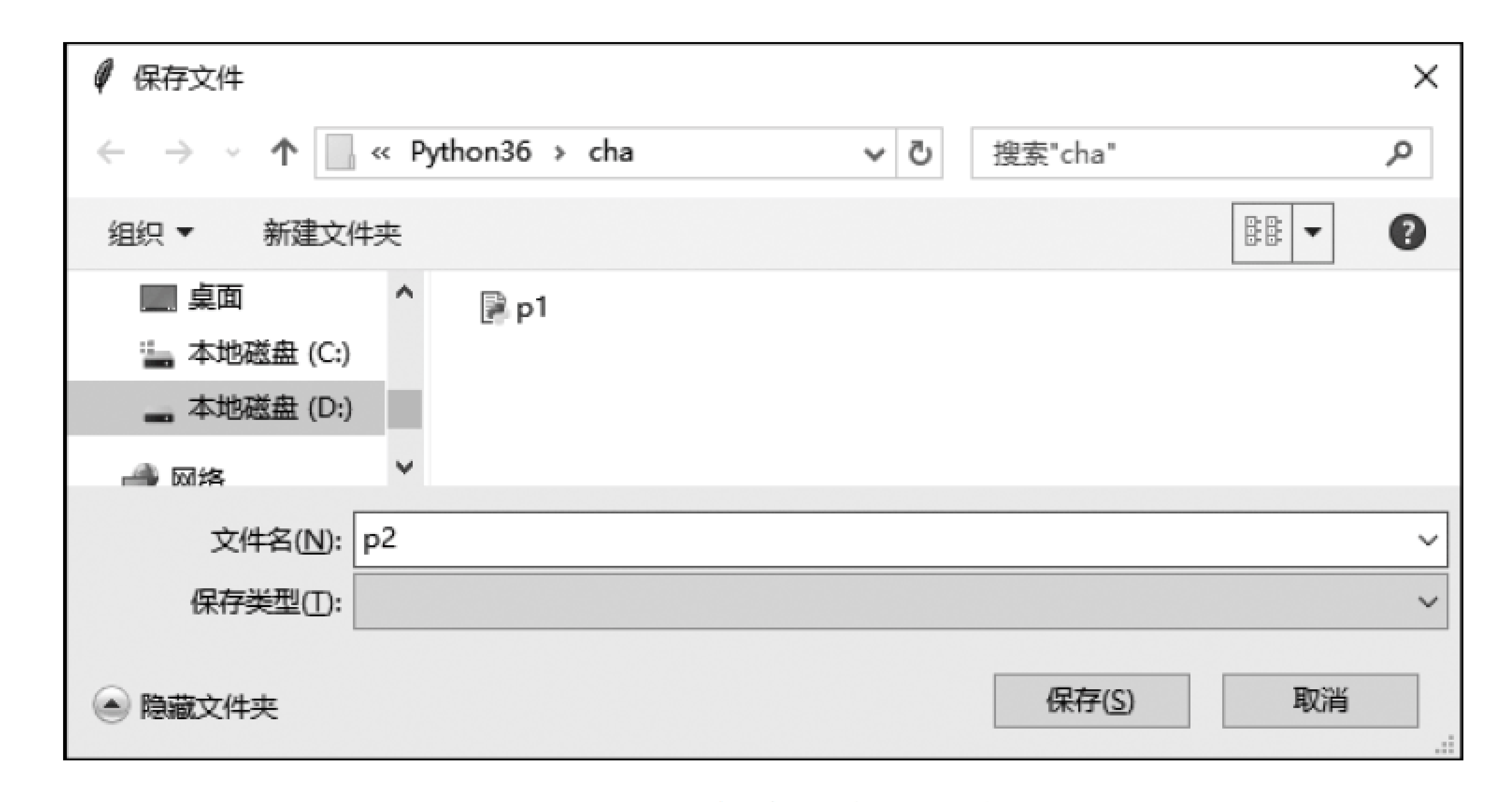


图 10-21　保存文件对话框

从图 10-20 和图 10-21 可以发现，这里的两个对话框是通过组件调用 Windows 系统得到的。换言之，这两个对话框并不是 Python 系统也不是 Tkinter 模块提供的。

2. 创建简单对话框

simpledialog(简单对话框)组件中的属性如表 10-14 所示。

表 10-14　simpledialog 组件的属性

参　数	说　明
title	指定简单对话框的标题
prompt	指定简单对话框中的显示文字
initialvalue	指定简单对话框中的初始值

【例 10-18】　simpledialog 组件示例。

源程序如下：

```
import tkinter
from tkinter import simpledialog
```

```
#定义函数实现整数的输入
def inputInt():
    r=simpledialog.askinteger("Python Tkinter","输入整数")
    print(r)
#定义函数实现实数的输入
def inputFloat():
    r=simpledialog.askfloat("Python Tkinter","输入实数")
    print(r)
#定义函数实现字符串的输入
def inputStr():
    r=simpledialog.askstring("Python Tkinter","输入字符串")
    print(r)
root=tkinter.Tk()
root.title("simpledialog 组件示例")
#设置 3 个按钮实现 3 种数据输入
btn_1=tkinter.Button(root,text="输入整数",command=inputInt)
btn_2=tkinter.Button(root,text="输入实数",command=inputFloat)
btn_3=tkinter.Button(root,text="输入字符串",command=inputStr)
#激活 3 个按钮
btn_1.pack(side="top")
btn_2.pack(side="top")
btn_3.pack(side="top")
root.mainloop()
```

运行结果如图 10-22 所示。

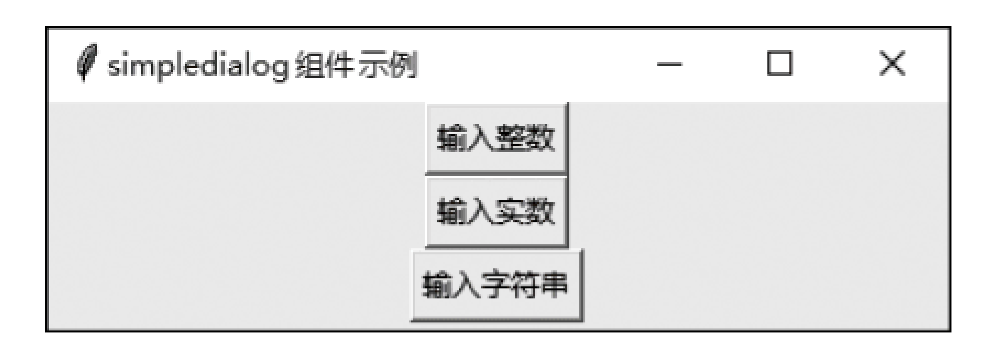


图 10-22　例 10-18 的运行结果 1

单击"输入整数"按钮,系统将显示如图 10-23 所示的对话框。

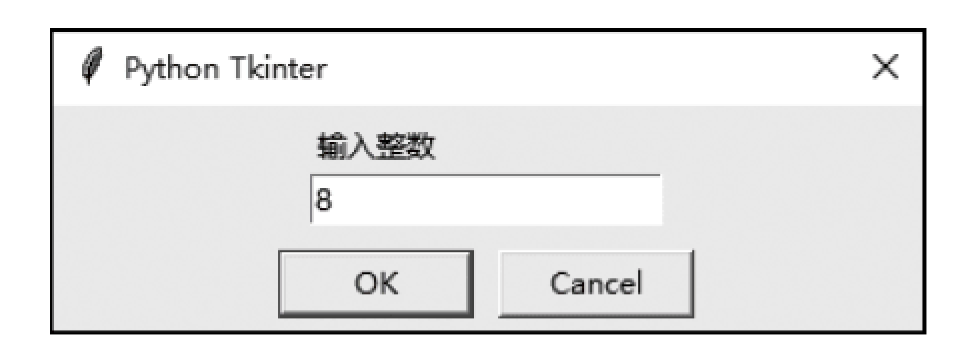


图 10-23　例 10-18 的运行结果 2

输入数据 8,则系统将显示该数;若没有输入就退出,则系统将显示 None,这是一个系统常量,表示函数没有返回值。

3. 创建消息框

messagebox(消息框)组件用于弹出提示框向用户进行告警,或让用户选择下一步如何操作。消息框包括很多类型,常用的有 info、warning、error、yesno、okcancel 等,分别对应不

同的图标、按钮以及弹出提示音。

【例 10-19】 messagebox 组件示例。

源程序如下：

```
import tkinter
#导入 messagebox 模块
from tkinter import messagebox
root=tkinter.Tk()
root.title("messagebox 组件示例")
def display():
    global n
    global btn_text
    n=n+1
    if n==1:
        messagebox.askokcancel("Python 程序设计","第 1 次显示")
        btn_text.set("第 1 次显示：Python 编程")
    elif n==2:
        messagebox.askquestion("Python 程序设计","第 2 次显示")
        btn_text.set("第 2 次显示：Python 编程")
    elif n==3:
        messagebox.askyesno("Python 程序设计","第 3 次显示")
        btn_text.set("第 3 次显示：Python 编程")
    else:
        n=0
        messagebox.showwarning("Python 程序设计","第 4 次显示")
        btn_text.set("第 4 次显示：Python 编程")
n=0
btn_text=tkinter.StringVar()
btn_text.set("单击")
btn=tkinter.Button(root,textvariable=btn_text,command=display)
btn.pack()
root.mainloop()
```

本例中的第 7 行设置全局变量 n，使函数和主程序均可以引用。

运行结果如图 10-24 所示。

若第 1 次单击按钮“单击”，系统将弹出如图 10-25 所示的消息框。



图 10-24　例 10-19 的运行结果 1

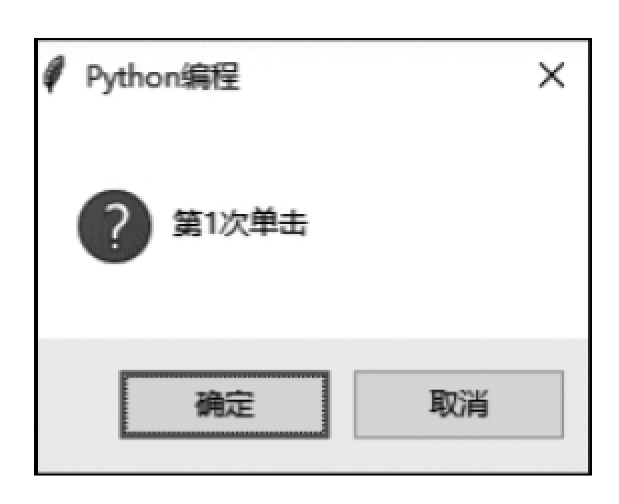


图 10-25　例 10-19 的运行与结果 2

分别可以选择“确定”“取消”和“关闭”操作。单击“确定”按钮，系统将弹出如图 10-26 所示的窗口，这将改变图 10-24 中的按钮文本。



图 10-26　例 10-19 的运行与结果 3

10.4　事件处理

前面介绍了按钮、输入框、框架、标签、列表框、菜单、滚动条、文本框、滑动杆、面板、对话框、消息框等图形界面对象，它们都可以直接接受用户的各种操作请求，使用过 Windows 系统的用户都非常熟悉这些界面对象及其操作。在 Python 程序中，可以在图形界面中实现各种人机交互操作。这些操作就是事件，而任何事件均是需要处理的，以便实现对应的操作请求。下面就介绍事件和事件处理。

10.4.1　事件类型

所谓事件(event)就是程序运行时发生的操作。例如，当用户敲击键盘上某一个键或是单击、移动鼠标等，Python 程序都应该对这些事件做出响应。Tkinter 模块中的组件通常能自行识别相应的事件，例如单击按钮、在输入框中输入字符、按键等事件。

事件可以分为如下 4 种类型：鼠标事件、键盘事件、窗口事件和响应事件。

1. 鼠标键盘事件

鼠标键盘事件如表 10-15 所示。

表 10-15　鼠标事件

事　　件	说　　明
<Button-1>	1 表示鼠标左键按下，2 表示中键按下，3 表示右键按下
<ButtonPress-1>	与<Button-1>相同
<ButtonRelease-1>	释放鼠标左键
<B1-Motion>	按住鼠标左键并移动
<Double-Button-1>	双击鼠标左键
<Enter>	鼠标指针进入某一组件区域
<Leave>	鼠标指针离开某一组件区域
<MouseWheel>	滚动鼠标滚轮

2. 键盘事件

键盘事件如表 10-16 所示。

表 10-16　键盘事件

事　　件	说　　明
<KeyPress-A>	按 A 键,A 可用其他键替代
<Alt-KeyPress-A>	同时按 Alt 和 A 键
<Double-KeyPress-A>	快速按两次 A 键,A 可用其他键替代
<Lock-KeyPress-A>	大写字母状态下按 A 键

任何键盘操作都将由硬件(键盘本身)的编码电路生成对应的 ASCII 码值,据此 Python 就可以进行事件处理。

3. 窗口事件

窗口事件如表 10-17 所示。

表 10-17　窗口事件

事　　件	说　　明
Activate	当组件由不可用状态时转为可用时触发
Configure	当组件大小改变时触发
Deactivate	当组件由可用状态转变为不可用时触发
Destroy	当组件被销毁时触发
Expose	当组件从被遮挡状态中暴露时触发
Unmap	当组件由显示状态变为隐藏状态时触发
Map	当组件由隐藏状态变为显示状态时触发
FocusIn	当组件获得焦点时触发
FocusOut	当组件失去焦点时触发
Property	当窗体的属性被删除或改变时触发
Visibility	当组件变为可视状态时触发

在选择框中进行选择或在输入框中输入字符时,Python 将其视为获得焦点并触发焦点事件;反之,若鼠标和键盘离开选择框或输入框时,则 Python 将失去焦点事件。例如,在输入电话号码时可以控制输入的必须是数字,若不是数字则禁止。

4. 响应事件

响应事件如表 10-18 所示。

事件类型必须由“<”和“>”来界定,它的一般引用格式如下:

```
[<modifier->]…<type>[<-detail>]
```

说明:

(1) <type>表示事件类型,例如鼠标单击、键盘按键等;

(2) <modifier>表示组合键定义,例如 Ctrl、Alt、Shift 键等;

表 10-18　响应事件

事　件	说　明	事　件	说　明
char	获取键盘的按键字符	widget	触发事件的对应组件
keycode	获取键盘的按键编码	width，heigh	获取组件的大小
keysym	获取键盘的按键符号	x，y	获取指定窗口中的鼠标位置
num	鼠标按键	x_root，y_root	获取整个屏幕中的鼠标位置
type	所触发的事件类型		

（3）＜detail＞表示按键或按鼠标方面的事件，例如 1 表示单击左键，2 表示单击中键，3 表示单击右键。

例如，＜Button-2＞表示单击鼠标中键，＜KeyPress-A＞表示按键盘中的 A 键，＜Control-Shift-KeyPress-C＞表示同时按 Ctrl、Shift 和 C 键。

10.4.2　事件绑定

触发事件后，程序可以使用事件处理函数来指定对触发事件所进行的操作。

事件由人机交互操作产生，而人机交互操作是在图形界面中的组件上实施的，所以程序需要建立处理指定事件的处理函数，这就是事件绑定。下面介绍对象绑定和标识绑定。

1. 对象绑定

在创建组件对象时，可以使用 command 属性来指定所需的事件处理函数。

【例 10-20】　对象绑定示例。

源程序如下：

```
import tkinter
from tkinter import Button
root=tkinter.Tk()
root.title("对象绑定示例")
#定义事件处理函数
def callback():
    print("为什么单击我?")
#由 Button 组件的 command 属性指定所调用的事件处理函数，即对象绑定
Bu1=tkinter.Button(root,text="设置事件处理",command=callback)
Bu1.pack()
root.mainloop()
```

运行结果如图 10-27 所示。

当单击“设置事件处理”按钮（第 9 行语句）时将由系统调用事件处理函数 callback()，由该函数在 IDLE 交互环境上显示：“为什么单击我?”。

图 10-27　例 10-20 的运行结果

2. 标识绑定

在 Canvas 画布中绘制各种图形，将图形与事件绑定可以使用标识绑定 tag_bind()函数。预先为图形定义标识 tag 后，通过标识 tag 来绑定

事件。例如：

```
cv.tag_bind(" root ", "<Button-1>", printRect)
```

这里的 cv 表示画布窗口，tag_bind 为标识绑定函数，它需要 3 个参数，第 1 个参数 root 表示触发事件的图形，第 2 个参数<Button-1>表示按鼠标左键，第 3 个参数 printRect 表示实现事件处理的函数。

【例 10-21】 标识绑定示例。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("标识绑定")
#定义 3 个函数分别实现 3 种事件处理
def printRect(event):
    print ("左键单击矩形边事件")
def printRect2(event):
    print ("右键单击矩形边事件")
def printLine(event):
    print ("左键单击线段事件")
#创建 Canvas 对象
cv=Canvas(root,bg="white")
rt1=cv.create_rectangle(10,10,120,120,width=3,tags="r1")
#绑定 item 与鼠标左键事件，即标识绑定
cv.tag_bind("r1","<Button-1>",printRect)
#绑定 item 与鼠标右键事件
cv.tag_bind("r1","<Button-3>",printRect2)
#创建一个 line, 并将其 tags 设置为"r2"
cv.create_line(200,80,320,80,width=3,tags="r2")
#绑定 item 与鼠标左键事件
cv.tag_bind("r2","<Button-1>",printLine)
cv.pack()
root.mainloop()
```

运行结果如图 10-28 所示。

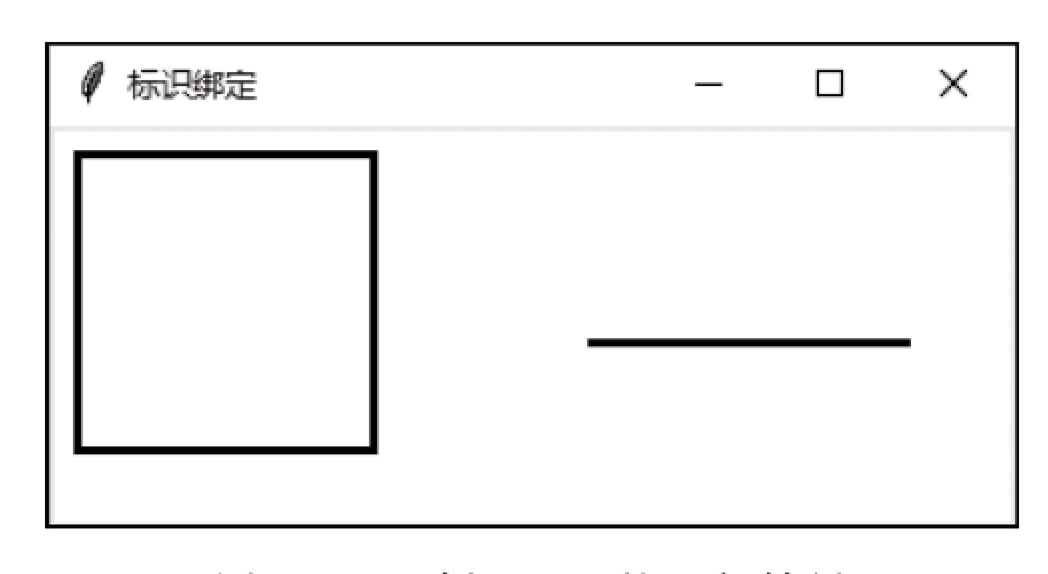


图 10-28　例 10-21 的运行结果

在图 10-28 中，进行鼠标操作将触发相应事件以及事件处理处理，全部过程均会同步显

示在 IDLE 交互环境中，如图 10-29 所示。

```
====================== RESTART: D:/Python36/cha/p1.py ======================
左键单击矩形边事件
左键单击线段事件
右键单击矩形边事件
```

图 10-29　标识绑定

10.4.3　键盘事件

【例 10-22】　键盘事件示例。

源程序如下：

```
from tkinter import *
#定义键盘事件处理的函数
def printkey(event):
    print("按键：  "+event.char)
root=Tk()
root.title("键盘事件")
#设置 Entry 输入框
entry=Entry(root)
#给 Entry 输入框绑定键盘事件<KeyPress>
#<KeyPress-x>监听键盘，如果为大写 A 即<KeyPress-A>，回车即<KeyPress-Enter>
entry.bind("<KeyPress>",printkey)
entry.pack()
root.mainloop()
```

运行结果如图 10-30 所示。

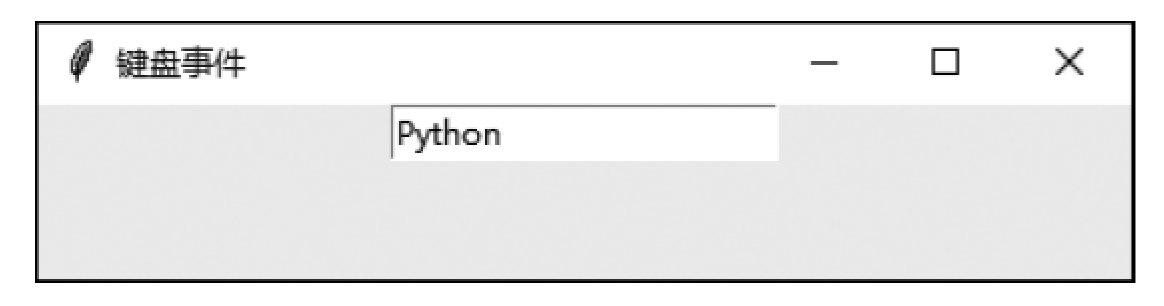
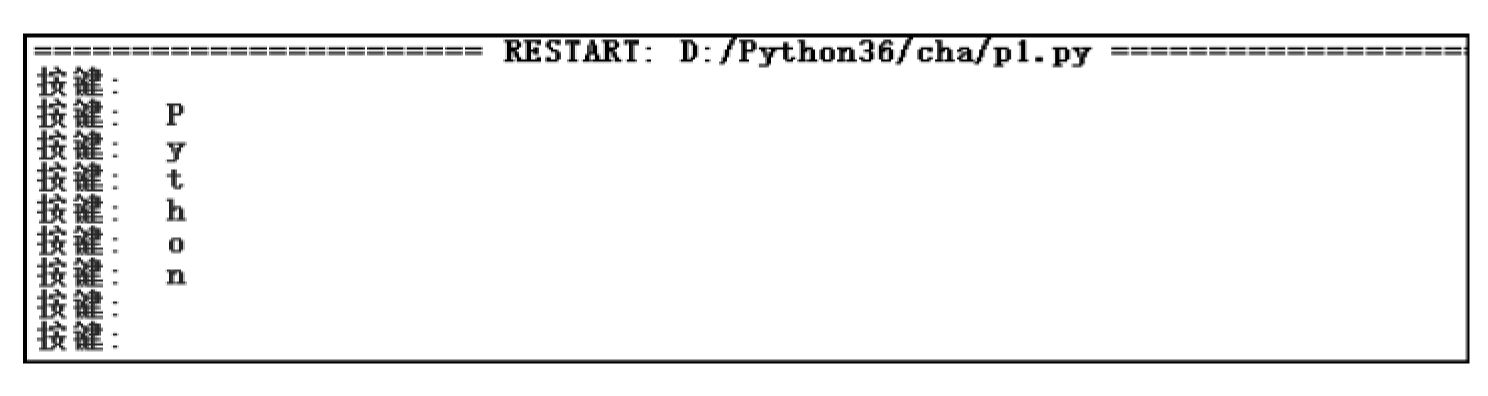


图 10-30　例 10-22 的运行结果

在图 10-30 中，要输入大写字母 P 需要按 Shift 键，最后要终止运行程序需要同时按 Ctrl 键和 C 键，均会触发键盘事件，所以共产生 3 个没有对应字符的“按键：”信息，如图 10-31 所示。



```
====================== RESTART: D:/Python36/cha/p1.py ==================
按键：
按键：  P
按键：  y
按键：  t
按键：  h
按键：  o
按键：  n
按键：
按键：
```

图 10-31　键盘事件

习　题　10

一、简答题

1. 什么是图形界面? 什么是图形界面设计?
2. Tkinter 模块的主要功能是什么?
3. Tkinter 模块主要有哪些组件?
4. 如何创建一个图形界面窗口?
5. 什么是布局管理? Python 有哪些布局管理?
6. Label 组件有哪些函数和属性?
7. Button 组件有哪些函数和属性?
8. Entry 和 Text 组件各有哪些函数和属性?
9. Listbox 组件有哪些函数和属性?
10. Radiobutton 和 Checkbutton 组件有哪些函数和属性?
11. Frame 组件有哪些函数和属性?
12. Scrollbar 组件有哪些函数和属性?
13. Menu 组件有哪些函数和属性?
14. 3 种类型的对话框各有哪些函数和属性?
15. 什么是事件处理?
16. 简述 Python 中的事件分类情况。
17. 如何实现事件的对象绑定。
18. 如何实现事件的标识绑定。

二、编程题

1. 使用 pack 布局,从上到下排列 3 个按钮(Button)。
2. 使用 grid 布局,用 9 个按钮(Button)表示如图 10-32 所示的网格。

1	2	3
8	9	4
7	6	5

图 10-32　按钮布局

3. 设计一个由 Entry 和一个 Button 组件构成的窗口,单击 Button 组件后,将使单行 Entry 组件中的输入字符串转换为全大写形式。

4. 设置 Text 组件为 3 行 36 列,编程将字符串“Python 程序设计”和“计算思维视角”放入其中。

5. 设置 Listbox 组件并初始化为 3 个城市名称,由单击不同的 3 个 Button 组件控制显示相应的城市名称。

6. 利用 Radiobutton 组件的 text 属性分别标注为 A、B、C、D,用于模拟单项选择题。

7. 用 Checkbutton 组件的 text 属性分别标注为 A、B、C、D 和 E，用于模拟多项选择题。

8. 编程实现在单击 Button 组件后，弹出消息框显示“欢迎学习 Python 程序设计”。

9. 编程：从 Text 组件中获得一个角度的度数，单击按钮后由事件处理函数将度数转换为弧度数。

第 11 章　绘 制 曲 线

在图形界面设计中，除 Tkinter 模块的组件外，Python 还提供许多绘制曲线的方法和显示技术。本章首先介绍 Canvas（画布）组件，以及如何用它绘制直线、矩形、多边形、圆弧、椭圆，显示位图、图像与文本，控制与变换图形；接着介绍 Python 内置的海龟程序和海龟绘图方法；最后介绍 3 个分形（fractal）图形和两种显示字体的方法。

11.1　Canvas 组 件

由对象从属上划分，Canvas（画布）组件是 Tkinter 模块的一个组件。

Canvas（画布）是一个矩形区域，在其中可以实现图形绘制和界面设计，例如在画布上可以绘制图形、设置文本与字体、安排组件、设计框架等。

11.1.1　Canvas 对象及其通用属性

创建 Canvas 对象的一般格式如下：

```
w=Canvas(<master>, <option>=value, …)
```

说明：

（1）w：所创建 Canvas 对象的实例名。

（2）<master>：表示 Canvas 对象的父窗口。

（3）<options>：有关 Canvas 对象的通用属性，如表 11-1 所示。

表 11-1　Canvas 对象的通用属性

属　　性	说　　明
bd	设置边框宽度，默认值是两个像素
bg	设置背景颜色
confine	设置画布是否滚动
cursor	在画布上使用的光标模式，例如箭头、圆圈、圆点等
height	设置画布高度
highlightcolor	设置高亮显示颜色
relief	设置边框类型，例如 sunken、raised、groove 和 ridge
scrollregion	由元组（w、n、e、s）定义画布滚动区域，分别表示左侧、顶部、右侧和底部
width	设置画布宽度
xscrollincrement	设置水平滚动的增量
xscrollcommand	若画布可以水平滚动，则该属性由 set()函数设置

续表

属　　性	说　　明
yscrollincrement	设置垂直滚动的增量
yscrollcommand	若画布可以垂直滚动,则该属性由 set()函数设置

11.1.2　屏幕坐标

为控制信息的定位显示,使显示器能够按照指定的屏幕坐标(以像素为单位)位置显示信息。因此,需要确定对应的屏幕坐标。先将屏幕分成若干行和若干列,行表示屏幕的横向位置 y,列表示屏幕的纵向位置 x,行和列的交叉点就是屏幕坐标。在显示器中,每一个坐标位置都可以用于文字或图形的显示。

屏幕坐标的设置情况可以分为如下两种:图形显示方式和文本显示方式。

在图形显示方式下,若显示器坐标系统设定为全屏幕 900×1600(即屏幕分辨率),对行号从上到下进行编号为 0,1,2,3,4,…,899,对列号从左到右进行编号为 0,1,2,3,4,…,1599。如果由(x,y)用于描述屏幕坐标位置,则屏幕坐标位置如表 11-2 所示。

表 11-2　屏幕坐标位置

屏幕坐标	说　　明	屏幕坐标	说　　明
(0,0)	表示屏幕左上角	(1599,0)	表示屏幕右上角
(0,899)	表示屏幕左下角	(1599,899)	表示屏幕右下角

文本显示方式与图形显示方式的表示原理是一样的,但现在极少用于界面设计。

11.2　绘制图形

Canvas 可用于绘制各种图形对象,通常由调用相关的图形绘制函数来实现。常用图形绘制函数如表 11-3 所示。

表 11-3　图形绘制函数

函　　数	说　　明	函　　数	说　　明
create_arc()	绘制圆弧	create_line()	绘制直线
create_oval()	绘制椭圆	create_polygon()	绘制多边形
create_rectangle()	绘制矩形		

11.2.1　绘制直线、矩形和多边形

1. 绘制直线

绘制直线可以使用 create_line()函数,其一般调用格式如下:

```
create_line(<直线起点坐标>, <直线终点坐标>, <选项>)
```

说明：

(1) ＜选项＞用于辅助定义绘制直线的效果，若为＜file＞，则用于指定填充颜色；若为＜width＞，则用于指定线条宽度；若为＜dash＞，则用于指定点虚线。

(2) 选项＜arrow＞用于指定箭头，例如 none 表示没有箭头；若为 first，则表示向左箭头；若为 last，则表示向右箭头；若为 both，则表示双向箭头。

【例 11-1】 使用 create_line()函数绘制直线。

源程序如下：

```
导入 tkinter 模块
from tkinter import *
root=Tk()
#设置窗口标题
root.title("绘制直线")
#创建 Canvas 对象，并设置背景颜色和窗口尺寸
cv=Canvas(root,bg="white",width=240,height=120)
#绘制没有箭头的直线,起点坐标(10,10),终点坐标(240,10)
cv.create_line(10,10,240,10,arrow="none")
#绘制有向左箭头的直线
cv.create_line(10,30,240,30,arrow="first")
#绘制有向右箭头的直线
cv.create_line(10,50,240,50,arrow="last")
#绘制有双向箭头的直线
cv.create_line(10,70,240,70,arrow="both")
#绘制倾斜的点虚线
cv.create_line(10,80,240,100,width=3,dash=7)
#激活 Canvas 对象
cv.pack()
root.mainloop()
```

运行结果如图 11-1 所示。

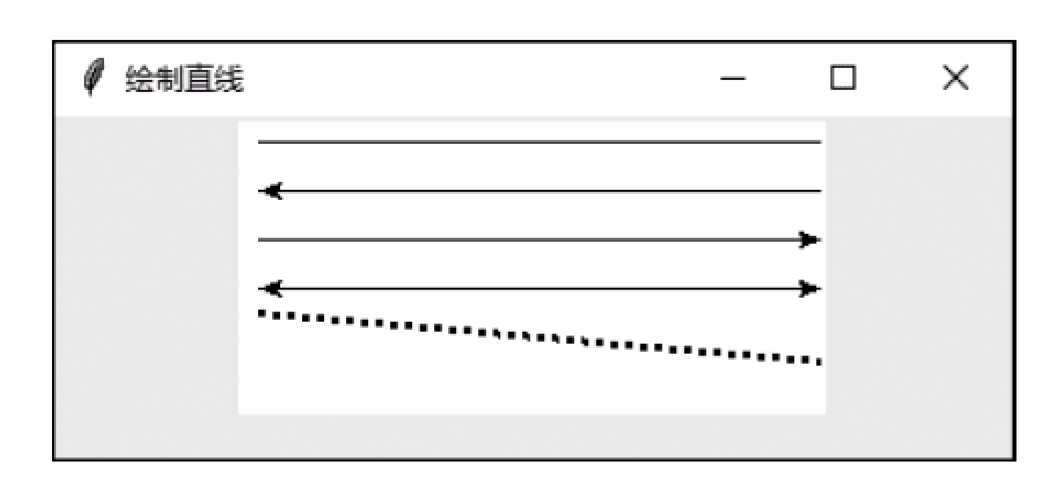


图 11-1　例 11-1 的运行结果

本例中的倾斜直线(第 17 行)是由起点坐标与终点坐标指定的，而且绘制直线是由两个坐标值确定的，并没有何为起点可为终点的顺序要求。

2. 绘制矩形

绘制矩形可以由 4 条直线实现，但效率过低。在 Python 语言中，绘制矩形可以使用 create_rectangle()函数，其一般调用格式如下：

```
create_rectangle(<矩形左上角坐标>, <矩形右下角坐标>, <选项>)
```

说明：＜选项＞用于辅助定义绘制矩形的效果。其中，＜file＞用于指定填充颜色，＜width＞用于指定线条宽度，＜dash＞用于指定点虚线，＜outline＞用于指定线条颜色，＜stipple＞用于指定画刷填充矩形。

【例 11-2】 使用 create_rectangle()函数绘制矩形。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("绘制矩形")
cv=Canvas(root,bg='white',width=320,height=140)
#绘制线宽为 3 个像素的矩形，并用灰色填充，左上角坐标(10,10)，右下角坐标(120,120)
cv.create_rectangle(10,10,120,120,width=3,fill="gray")
#绘制边框为 3 个像素且红色的矩形
cv.create_rectangle(140,10,320,120,width=3,outline="red")
cv.pack()
root.mainloop()
```

运行结果如图 11-2 所示。

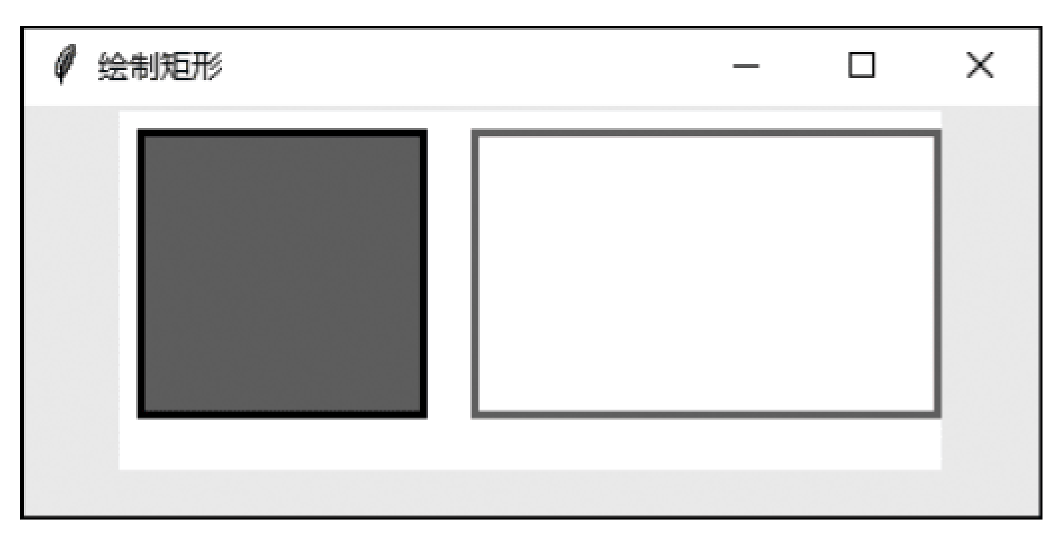


图 11-2　例 11-2 的运行结果

本例中的第一个矩形是正方形，由矩形的左上角和右下角坐标定义而得，Python 并没有提供专门绘制正方形的函数。

3. 绘制多边形

绘制多边形可以使用多条直线连线而成，只是计算每条直线的端点坐标很麻烦。在 Python 中，绘制多边形可以使用 create_polygon()函数，其一般调用格式如下：

```
create_polygon(<多边形坐标列表>, <选项>)
```

说明：

(1) ＜多边形坐标列表＞用于指定以顺时针方向表示的多边形端点坐标。

(2) ＜选项＞用于辅助定义绘制多边形的效果。其中，＜file＞用于指定填充颜色，＜width＞用于指定线条宽度，＜dash＞用于指定点虚线，＜outline＞用于指定线条颜色。

(3) 选项＜smooth＞用于指定多边形是否平滑连线，值为 0 表示折线，值为 1 表示平滑曲线。

【例 11-3】 使用 create_polygon()函数绘制多边形。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("绘制多边形")
cv=Canvas(root,bg="white",width=240,height=120)
#绘制红色边框且灰色填充的等腰三角形
cv.create_polygon(35,10,10,60,60,60,outline="red",fill="gray")
#绘制线宽为 3 个像素、用黑色填充且蓝色边框的直角三角形
cv.create_polygon(70,10,120,10,120,60,outline="blue",width=3,file="black")
#绘制黑色填充的正方形
cv.create_polygon(130,10,180,10,180,60,130,60,file="black")
#绘制线宽为 3 个像素且红色边框的两个垂直对称三角形
cv.create_polygon(190,10,240,10,190,60,240,60,fill="white",outline="red",width
=3)
cv.pack()
root.mainloop()
```

运行结果如图 11-3 所示。

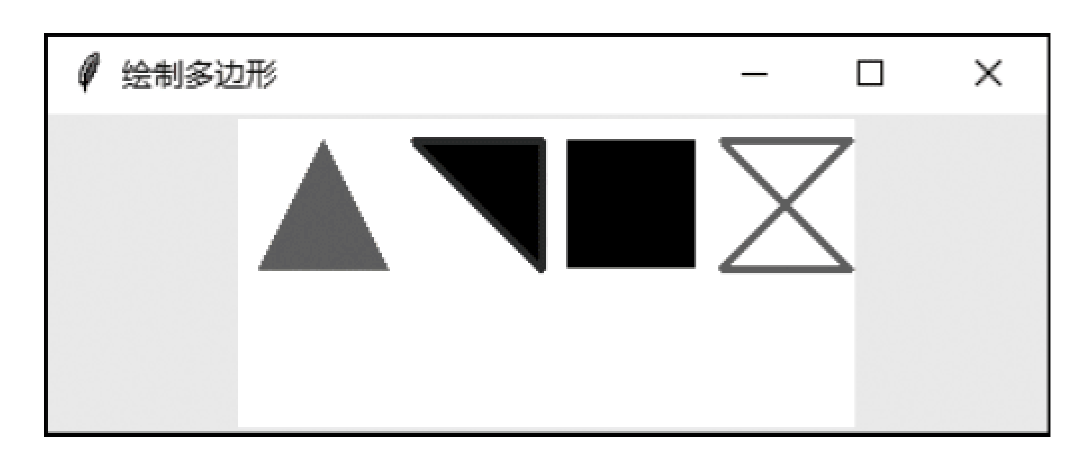


图 11-3　例 11-3 的运行结果

本例中的三角形和正方形均是以多边形结构实现的。

11.2.2　绘制圆弧和椭圆

1. 绘制圆弧

绘制圆弧可以使用 create_arc()函数，其一般调用格式如下：

```
create_arc(<圆弧框线左侧坐标>, <圆弧框线右侧坐标>, <选项>)
```

说明：

(1) ＜坐标＞形式是 x，y，例如 10，10。

(2) ＜选项＞用于辅助定义绘制图形的效果。其中，＜file＞用于指定填充颜色，＜width＞用于指定线条宽度，＜start＞用于指定起始角度，＜extent＞用于指定角度偏移量。

(3) 选项＜style＞用于指定圆弧类型，例如 pieslice 表示扇形，chord 表示弓形，arc 表示圆弧线。

【例 11-4】　调用 create_arc()函数绘制圆弧。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("绘制圆弧")
cv=Canvas(root,bg="gray",height=250,width=300)
#设置两个坐标,由无圆括号的元组表示
coord=10,50,240,210
#绘制圆弧并设置白色填充
cv.create_arc(coord,start=0,extent=150,fill="white")
cv.pack()
root.mainloop()
```

运行结果如图 11-4 所示。

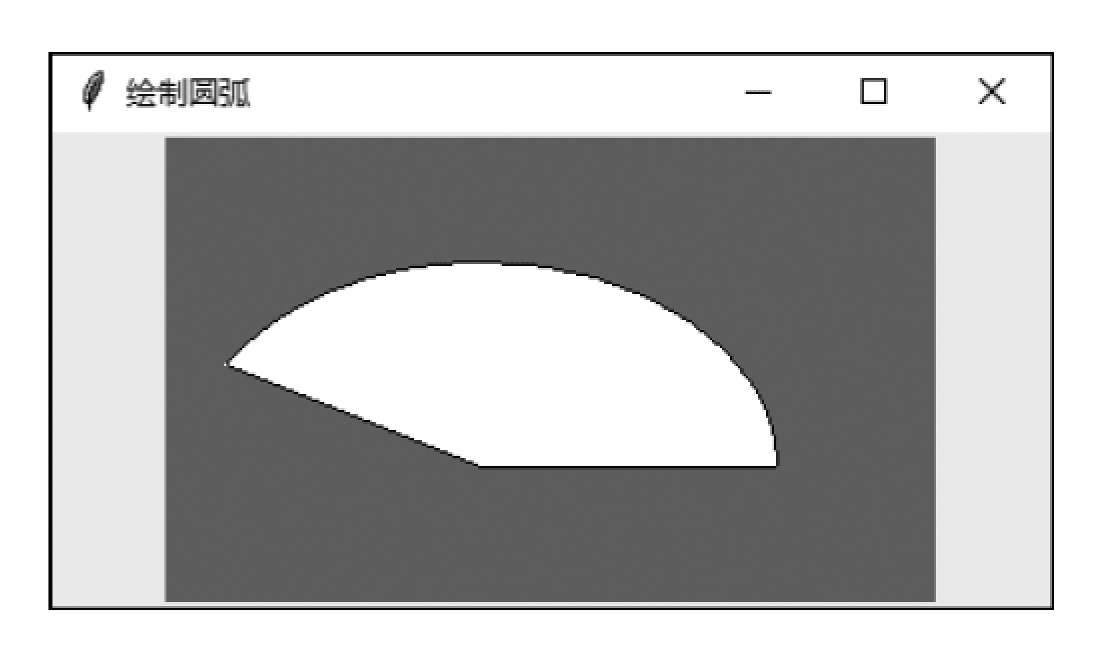


图 11-4　例 11-4 的运行结果

【例 11-5】　调用 create_arc()函数绘制扇形线、弓形线和圆弧线。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("绘制扇形线、弓形线和圆弧线")
cv=Canvas(root,bg="white")
#使用默认参数绘制扇形线
cv.create_arc((10,10,120,120),)
#按指定类型绘制扇形线
cv.create_arc((10,10+70,120,120+70),style="pieslice")
#按指定类型绘制弓形线
cv.create_arc((10,10+140,120,120+140),style="chord")
#按指定类型绘制圆弧线
cv.create_arc((10,10+210,120,120+210),style="arc")
#设置起始角度(<start>)和角度偏移量(<extent>, 以反时针方向)绘制弧线
cv.create_arc((140,140,240,240),start=10,extent=110)
cv.pack()
root.mainloop()
```

运行结果如图 11-5 所示。

2. 绘制椭圆

绘制椭圆可以使用 create_oval()函数,其一般调用格式如下：

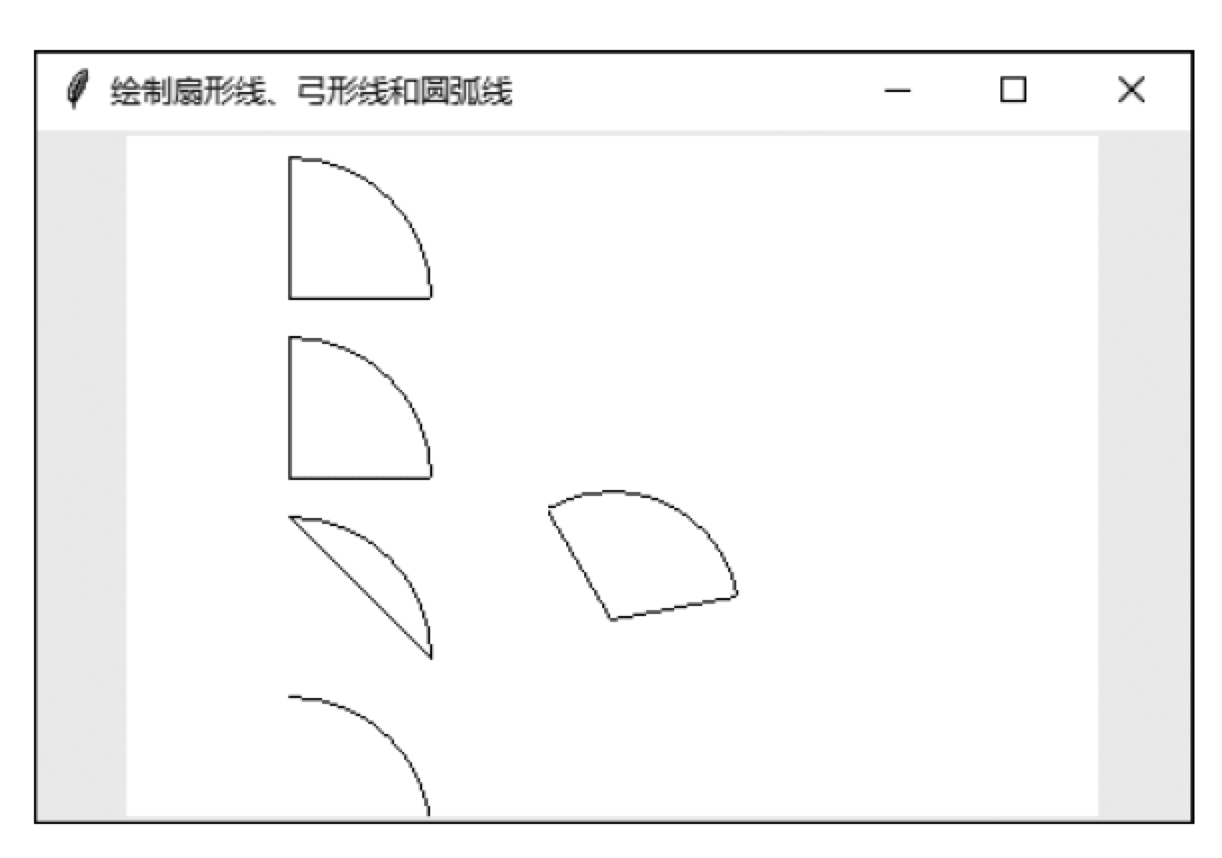


图 11-5　例 11-5 的运行结果

```
create_oval(<椭圆左侧坐标>, <椭圆右侧坐标>, <选项>)
```

说明：

(1) ＜椭圆左侧坐标＞和＜椭圆右侧坐标＞用于指定椭圆的两个顶点坐标。

(2) ＜选项＞用于辅助定义绘制椭圆的效果，若为＜file＞则指定填充颜色，若为＜width＞，则指定线条宽度，若为＜outline＞，则指定线条颜色。

【例 11-6】 使用 create_oval()函数绘制椭圆和圆。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("绘制椭圆和圆")
cv=Canvas(root,bg="white",width=200,height=100)
#以两个像素的红色线宽且灰色填充绘制椭圆
cv.create_oval(10,10,100,50,outline="red",fill="gray",width=2)
#以两个像素的蓝色线宽且白色填充绘制正圆
cv.create_oval(100,10,190,100,outline="blue",fill="white",width=2)
cv.pack()
root.mainloop()
```

运行结果如图 11-6 所示。

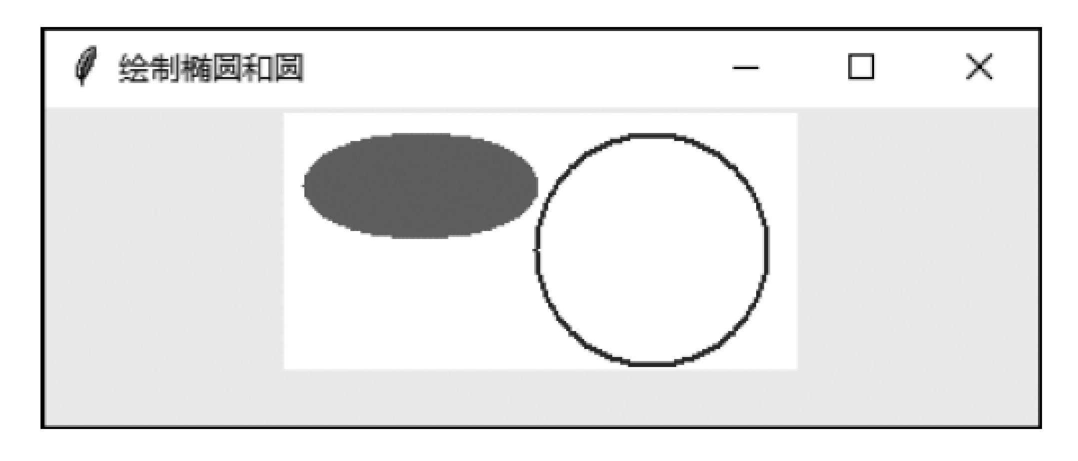


图 11-6　例 11-6 的运行结果

本例中的第 7 行使用椭圆函数绘制正圆是由设置图形的宽度与高度相同实现的。

11.3　显示位图、图像与文本

为了加强视觉感知效果，计算机可提供许多图像文件格式供程序员自由选择。例如，Python 中使用的没有经过压缩的 BMP 格式文件和经过压缩的 GIF 格式文件。其中，位图(Bitmap，BMP)文件是由点阵构成的，即由许多称作像素的点组成。这些点可以进行各种排列和染色来构成图像。由于这种文件没有压缩，所以具有通用性，但它需要占用很多的存储空间；GIF 格式在各种图像处理软件中得到应用，属于压缩的文件格式，因此占用存储空间较小，适合于进行网络传输和程序处理。

若读者所用的图像文件不是 BMP 格式和 GIF 格式，则只有将图像文件格式进行转换。

11.3.1　显示位图

显示位图可以使用 create_bitmap()函数，其一般调用格式如下：

```
create_bitmap(<位图显示的左上角坐标>, <bitmap>=位图字符串, <选项>)
```

说明：

(1) ＜位图显示的左上角坐标＞用于指定显示位图的起始位置。

(2) ＜bitmap＞用于指定位图文件(BMP 格式)。

(3) ＜选项＞用于辅助定义显示位图的效果，其中＜activebitmap＞表示激活位图文件，＜disablebitmap＞表示禁用位图文件。

【例 11-7】　使用 create_bitmap()函数显示位图。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("显示位图")
cv=Canvas(root)
#初始化字典, 分别表示 7 个 BMP 格式的文件
d={1:"info",2:"hourglass",3:"questhead",4:"warning",5:"gray12",6:"gray50",7:
"gray75"}
#使用字典中的样式, 分别显示 7 个位图文件
for i in d:
    cv.create_bitmap((36*i,24),bitmap=d[i])
cv.pack()
root.mainloop()
```

运行结果如图 11-7 所示。

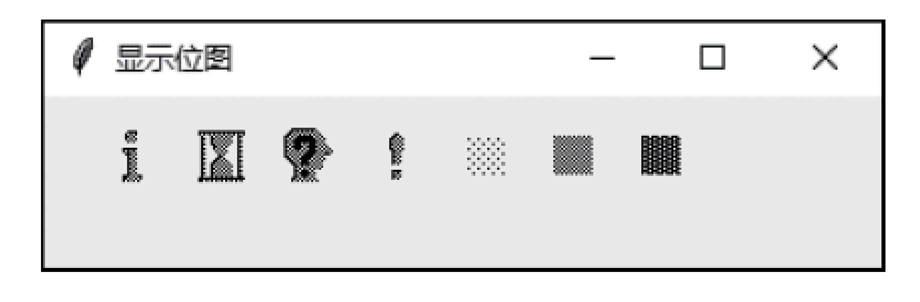


图 11-7　例 11-7 的运行结果

本例中的第 6 行指定的位图文件是 Python 系统内置的，即可以直接引用。

11.3.2　显示图像

正如显示位图必须是 BMP 图像文件，那么显示图像则必须是 GIF 图像文件，所用 create_image()函数的一般调用格式如下：

```
create_image(<图像显示的左上角坐标>, <image>=图像字符串, <选项>)
```

说明：

(1) <图像显示的左上角坐标>用于指定显示图像的起始位置。

(2) <image>用于指定图像文件(GIF 格式)。

(3) <选项>用于辅助定义显示图像的效果，其中<activeimage>表示激活图像文件，<disableimage>表示禁用图像文件。

【例 11-8】 使用 create_image()函数显示图像。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("显示图像")
cv=Canvas(root)
#定义 4 个 GIF 图像文件
img1=PhotoImage(file="img1.gif")
img2=PhotoImage(file="img2.gif")
img3=PhotoImage(file="img3.gif")
img4=PhotoImage(file="img4.gif")
#分别定位显示 4 个图像文件
cv.create_image((40,100),image=img1)
cv.create_image((140,100),image=img2)
cv.create_image((240,100),image=img3)
cv.create_image((340,100),image=img4)
cv.pack()
root.mainloop()
```

运行结果如图 11-8 所示。

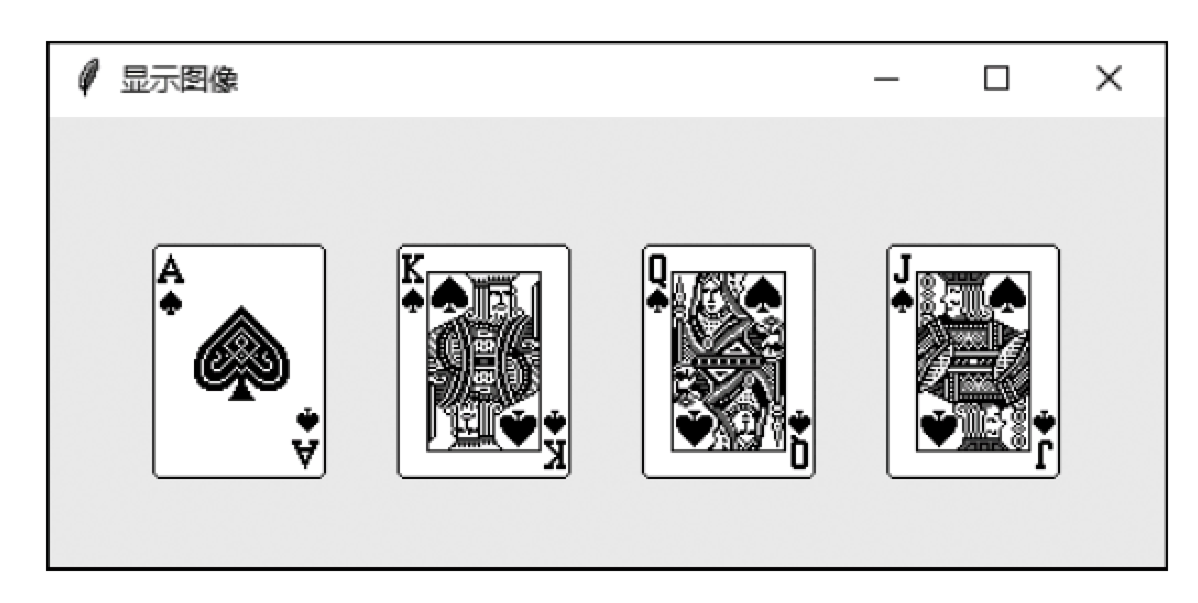


图 11-8　例 11-8 的运行结果

11.3.3 显示文本

显示文本可以使用 create_text()函数,其一般调用格式如下:

```
create_text(<显示文本的左上角坐标>, <text>, <file>, <anchor>, <justify>)
```

说明:

(1) <显示文本的左上角坐标>:用于指定显示文本的起始位置。

(2) <text>:用于指定文本。

(3) <fill>:用于指定文本颜色。

(4) <anchor>:用于指定文字对象的位置,例如<w>表示左对齐,<e>表示右对齐,<n>表示顶端对齐,<s>表示底端对齐,<nw>表示左上对齐,<sw>表示左下对齐,<se>表示右下对齐,<ne>表示右上对齐,<center>表示居中对齐。

(5) <justify>:用于指定文字对象中的文字位置,<left>表示左对齐,<right>表示右对齐,<center>表示居中对齐。

【例 11-9】 使用 create_text()函数显示文本。

源程序如下:

```
from tkinter import *
root=Tk()
root.title("显示文本")
cv=Canvas(root,bg="white",width=240,height=120)
#以左上对齐方式定位显示文本
cv.create_text((20,10),text="Python 编程",fill="blue",anchor="nw")
#以右下对齐方式定位显示文本
cv.create_text((120,60),text="计算思维视角",fill="blue",anchor="se")
#以右下对齐方式定位显示文本
cv.create_text((200,120),text="清华大学出版社",fill="blue",anchor="se")
cv.pack()
root.mainloop()
```

运行结果如图 11-9 所示。



图 11-9 例 11-9 的运行结果

注意: 本例中的定位方式为两种,一是对齐定位,二是坐标定位。

11.4　控制图形

除绘制图形外，还可以进行图形控制，例如删除图形、修改图形、移动图形和缩放图形。常用控制图形(包含线条)的函数如表 11-4 所示。

表 11-4　控制图形函数

函　　数	说　　明
delete(＜图形对象＞)	删除图形
itemconfig(＜图形对象＞,＜位移量坐标＞,＜水平缩放比＞,＜垂直缩放比＞)	修改图形
move(＜图形对象＞,＜位移量坐标＞)	移动图像
coords(＜图形对象＞,＜左上角坐标＞,＜右下角坐标＞)	返回图形对象的位置坐标(元组数据)

11.4.1　删除图形

删除图形可以使用 delete()函数，其一般调用格式如下：

```
Canvas 对象.delete(<图形对象>)
```

说明：＜图形对象＞是与每一个绘制图形命令对应的，即一个窗口中可能含有许多图形对象。

【例 11-10】　使用 delete()函数删除图形。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("删除当前图形")
cv=Canvas(root,bg="white",width=240,height=120)
#绘制没有箭头的直线
rt=cv.create_line(10,10,240,10,arrow="none")
#删除没有箭头的直线
cv.delete(rt)
#显示删除图形后的效果
cv.create_text((10,40),text="已删除图形文件",fill="blue",anchor="nw")
cv.pack()
root.mainloop()
```

运行结果如图 11-10 所示。



图 11-10　例 11-10 的运行结果

本例中的删除图形操作是以图形对象为函数参数实现的，例如第 8 行的 cv.delete(rt)。

11.4.2 移动图形

移动图形可以使用 move()函数，其一般调用格式如下：

```
Canvas 对象.move(<图形对象>, <位移量坐标>)
```

说明：

(1) ＜图形对象＞是由每一个绘制图形命令对应的。

(2) ＜位移量坐标＞用于表示移动图形的偏移值，若为正值，则表示向右或向下移动；若为负值，则表示向左或向上移动，且不能超过窗口范围。

【例 11-11】 使用 move()函数移动图形。

源程序如下：

```python
from tkinter import *
root=Tk()
root.title("移动图形")
cv=Canvas(root,bg="white",width=200,height=120)
#绘制第一个矩形
rt1=cv.create_rectangle(20,20,160,120,outline="blue",width=2)
cv.pack()
#绘制第二个矩形与第一个矩形完全重合，只是框线颜色不同
rt2=cv.create_rectangle(20,20,160,120,outline="black",width=2)
#移动第一个矩形
cv.move(rt1,20,-10)
cv.pack()
root.mainloop()
```

本例中，首先使用 create_rectangle()函数绘制两个完全重合的矩形，然后在第 11 行中使用 move()函数指定移动坐标为(20，－10)，表示图形向右移动 20 个像素点，向上移动 10 个像素点。

运行结果如图 11-11 所示。

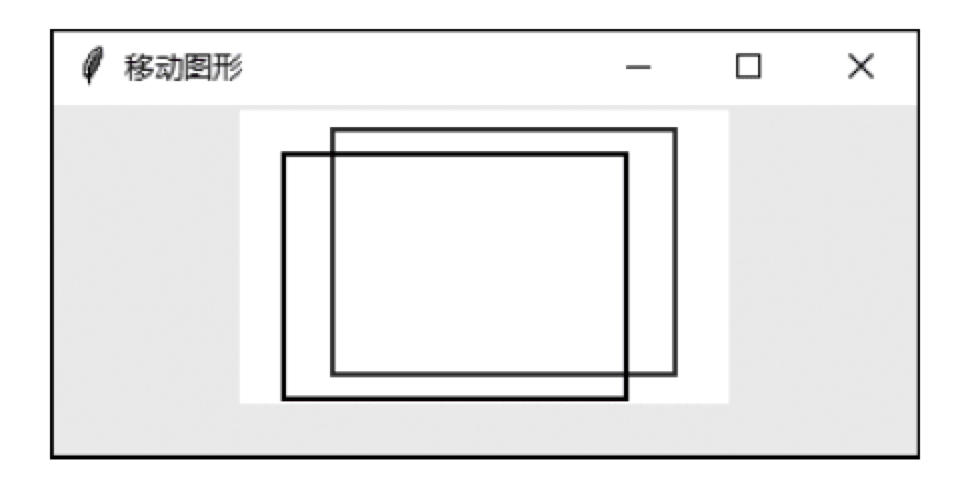


图 11-11 例 11-11 的运行结果

11.4.3 位置坐标

返回图形的位置坐标可以使用 coords()函数，其一般调用格式如下：

```
Canvas 对象.coords(<图形对象>, <左上角坐标>, <右下角坐标>)
```

说明：

(1) ＜图形对象＞是由每一个绘制图形命令对应的。

(2) ＜左上角坐标＞和＜右下角坐标＞用于指定图形的两个位置坐标。

【例 11-12】 使用 coords()函数返回图形的位置坐标。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("指定图形坐标")
cv=Canvas(root)
#定义 3 个图像文件
img1=PhotoImage(file="img5.gif")
img2=PhotoImage(file="img6.gif")
img3=PhotoImage(file="img7.gif")
#分别定位显示 3 个图像文件
rt1=cv.create_image((80,100),image=img1)
rt2=cv.create_image((180,100),image=img2)
rt3=cv.create_image((280,100),image=img3)
#重新设置第二个图像对象的坐标位置
cv.coords(rt2,(180,80))
#绘制第一个矩形
rt4=cv.create_rectangle(45,160,200,230,outline="blue",width=2)
#重新设置第一个矩形的坐标位置
cv.coords(rt4,(70,160,280,200))
cv.pack()
root.mainloop()
```

运行结果如图 11-12 所示。



图 11-12　例 11-12 的运行结果(移动前)

若将第 18 行的语句 cv.coords(rt4,(70,160,280,200))变为注释语句，则显示效果如图 11-13 所示。

11.4.4　缩放图形

缩放图形可以使用 scale()函数，其一般调用格式如下：

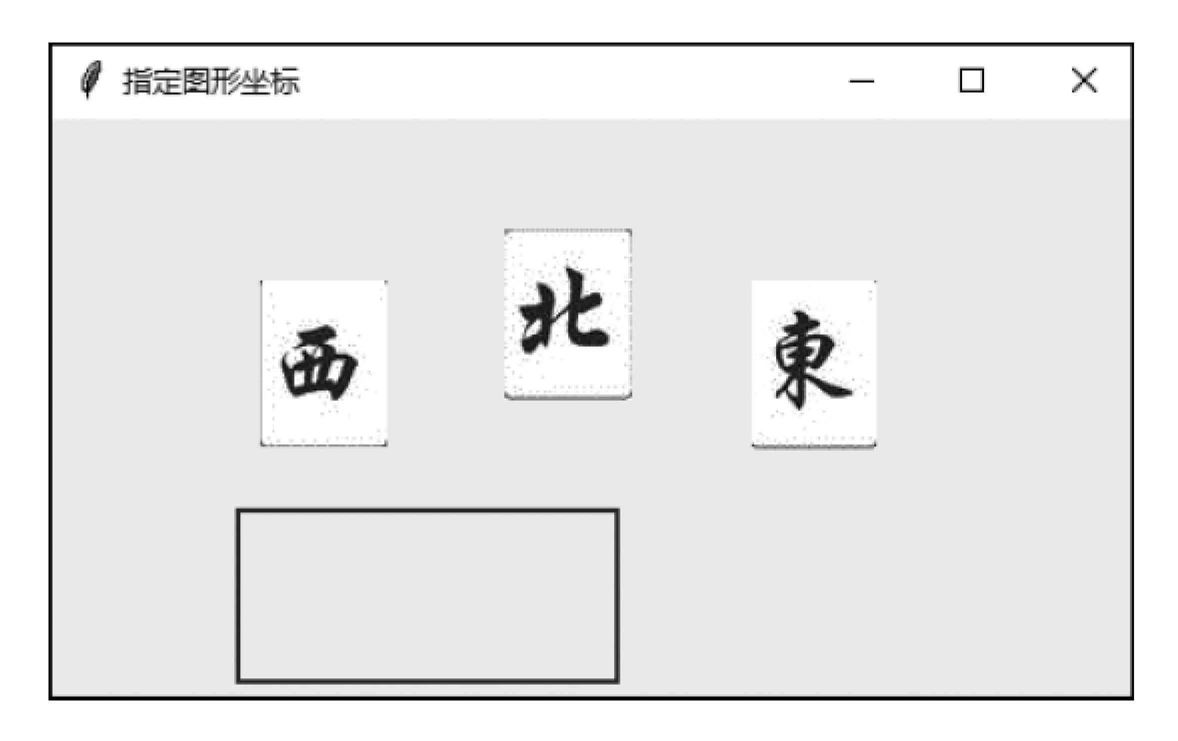


图 11-13　例 11-12 的运行结果(移动后)

```
Canvas 对象.scale(<图形对象>,<位移量坐标>,<水平缩放比>,<垂直缩放比>)
```

说明：

(1) ＜图形对象＞是由每一个绘制图形命令对应的。

(2) ＜位移量坐标＞用于表示移动图形的偏移值,若为正值,则表示向右或向下移动；若为负值,则表示向左或向上移动,且不能超过窗口范围。

(3) 必选项＜缩放比＞是一个正值,若大于 1 则表示放大,否则表示缩小。

【例 11-13】　使用 scale()函数缩放图形。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("缩放图形")
cv=Canvas(root,bg="white",width=240,height=180)
#绘制第一个矩形
rt1=cv.create_rectangle(10,10,110,110,outline="blue",width=2)
#绘制第二个矩形
rt2=cv.create_rectangle(20,20,120,120,outline="black",width=2)
#cv.scale(rt1,10,10,1.6,1)                          #水平方向放大 1.6 倍
#cv.scale(rt2,100,100,1,1.2)                        #垂直方向放大 1.2 倍
cv.pack()
root.mainloop()
```

运行结果如图 11-14 所示。

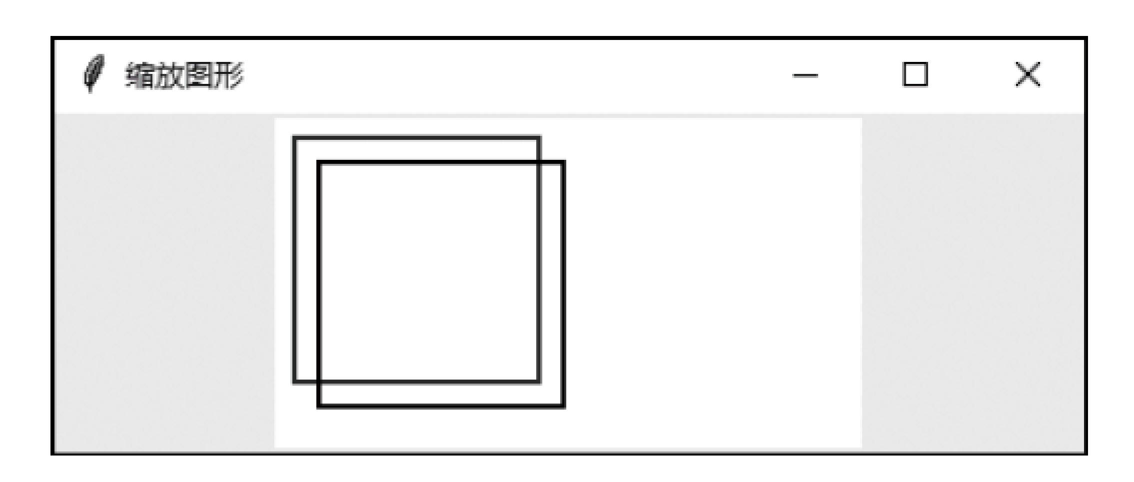


图 11-14　例 11-13 的运行结果(缩放前)

若将第 9、10 行的语句中的注释标记删除,则显示效果如图 11-15 所示。

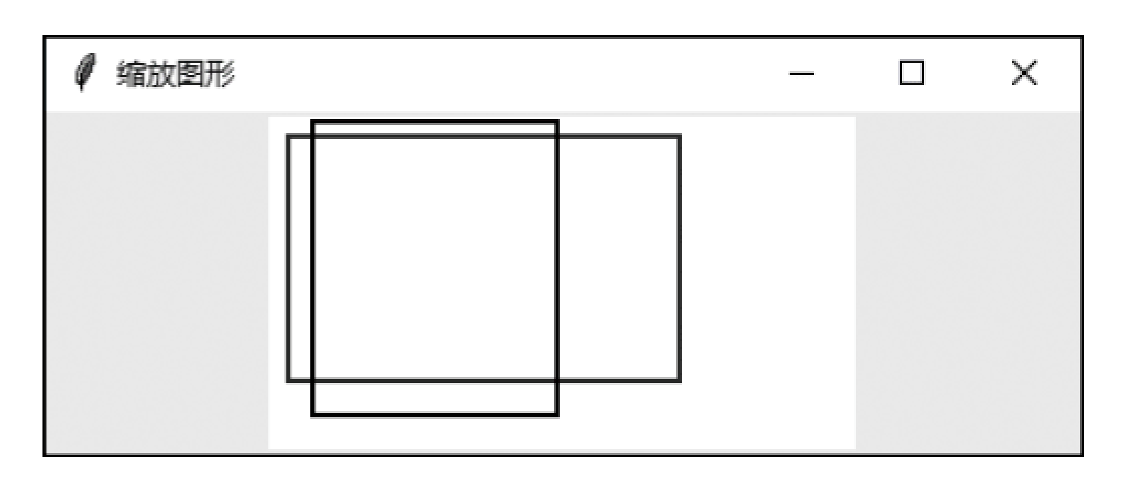


图 11-15　例 11-13 的运行结果(缩放后)

11.4.5　绘制函数图形

【例 11-14】 绘制正弦曲线。

源程序如下：

```
from tkinter import *
from math import sin,pi
#设置画布的宽度和高度
wth=510
hgt=210
#设置曲线的起点坐标
begin_x=2
begin_y=hgt/2
#设置 x 轴和 y 轴的缩放倍数
SCALE_X=40
SCALE_Y=100
#设置曲线的弧度范围
END_ARC=360 * 2
#设置函数图的起点坐标
o_x=0
o_y=0
x=0
y=0
#设置起始的弧度
arc=0
root=Tk()
root.title("绘制正弦曲线")
c=Canvas(root,bg="white",width=wth,height=hgt)
c.pack()
#绘制 X 轴和 Y 轴
c.create_line(begin_x,0,begin_x,hgt)
c.create_line(0,begin_y,wth,begin_y)
#绘制曲线
for i in range(0,END_ARC+1,10):
    arc=pi * i * 2/360
    x=begin_x+arc * SCALE_X
```

```
        y=begin_y-sin(arc) * SCALE_Y
        c.create_line(o_x,o_y,x,y)
        o_x=x
        o_y=y
    print("Over!")
```

运行结果如图 11-16 所示。

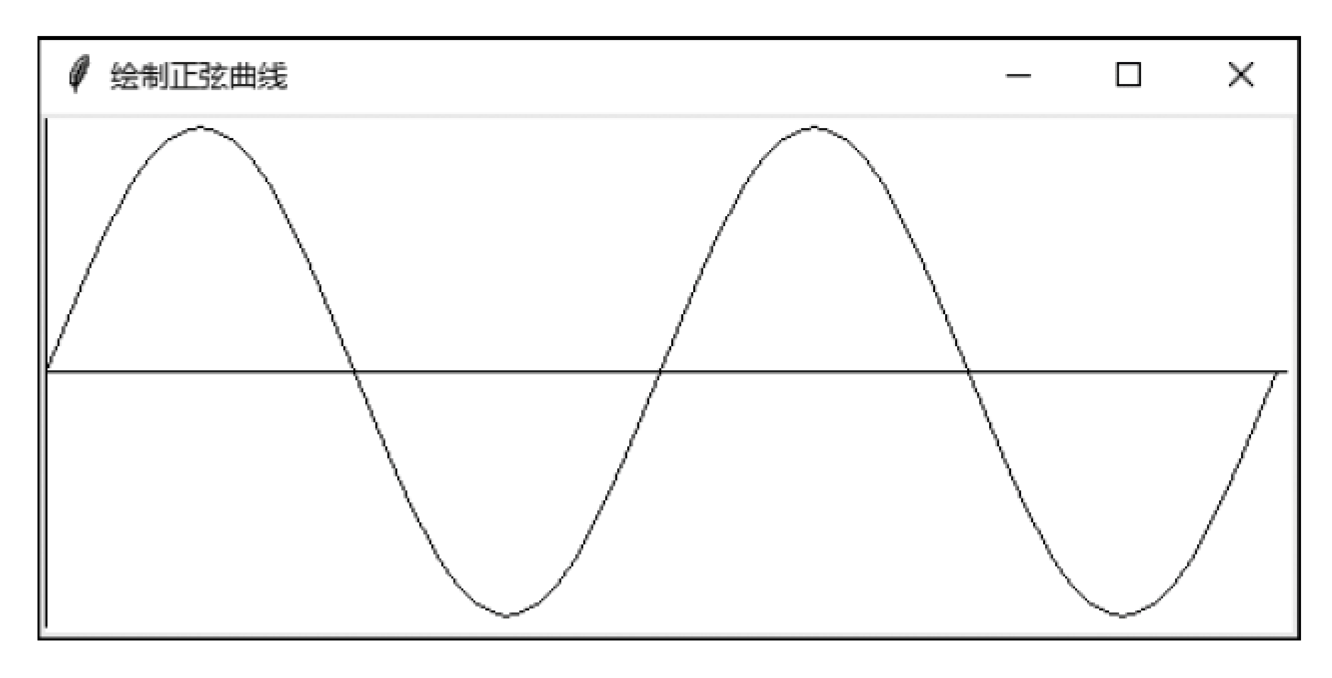


图 11-16　例 11-14 的运行结果

11.5　体验内置的 turtle 演示程序

Python 提供的 turtle 模块,以让程序员通过简单的程序创建直观且丰富的视觉效果。这里使用 turtle 一词,表示是在模拟一只海龟在屏幕上的移动过程。由于整个移动的过程与方向是完全可见的,显然与过去计算机高速生成一条曲线完全不同。

11.5.1　利用 IDLE 内置程序

利用 IDLE 交互环境中的 Turtle Demo 选项,可以调用 Python 系统中的 turtle 演示程序。

具体操作过程如下:

在 Windows 10 的"开始"菜单中选中 IDLE(Python 3.6 64-bit)选项,进入 IDLE 环境,在窗口中选中 Help 菜单选项,如图 11-17 所示。

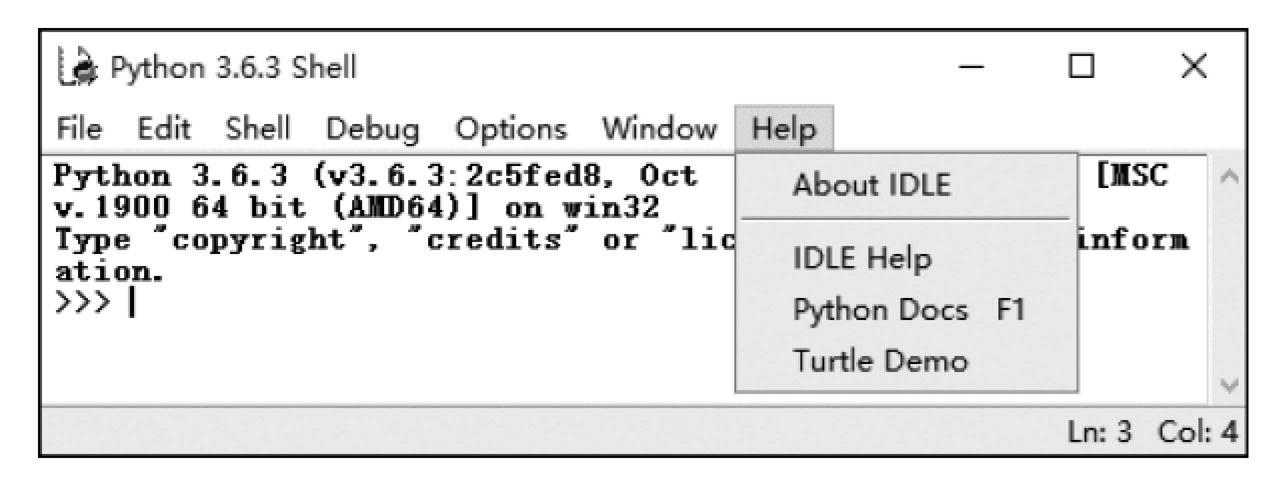


图 11-17　Help 菜单

选中 Turtle Demo 菜单选项,弹出如图 11-18 所示的窗口。

选中 Examples 菜单选项,其中列出 19 个示范的海龟程序,如图 11-19 所示。下面以时钟显示为例进行说明。

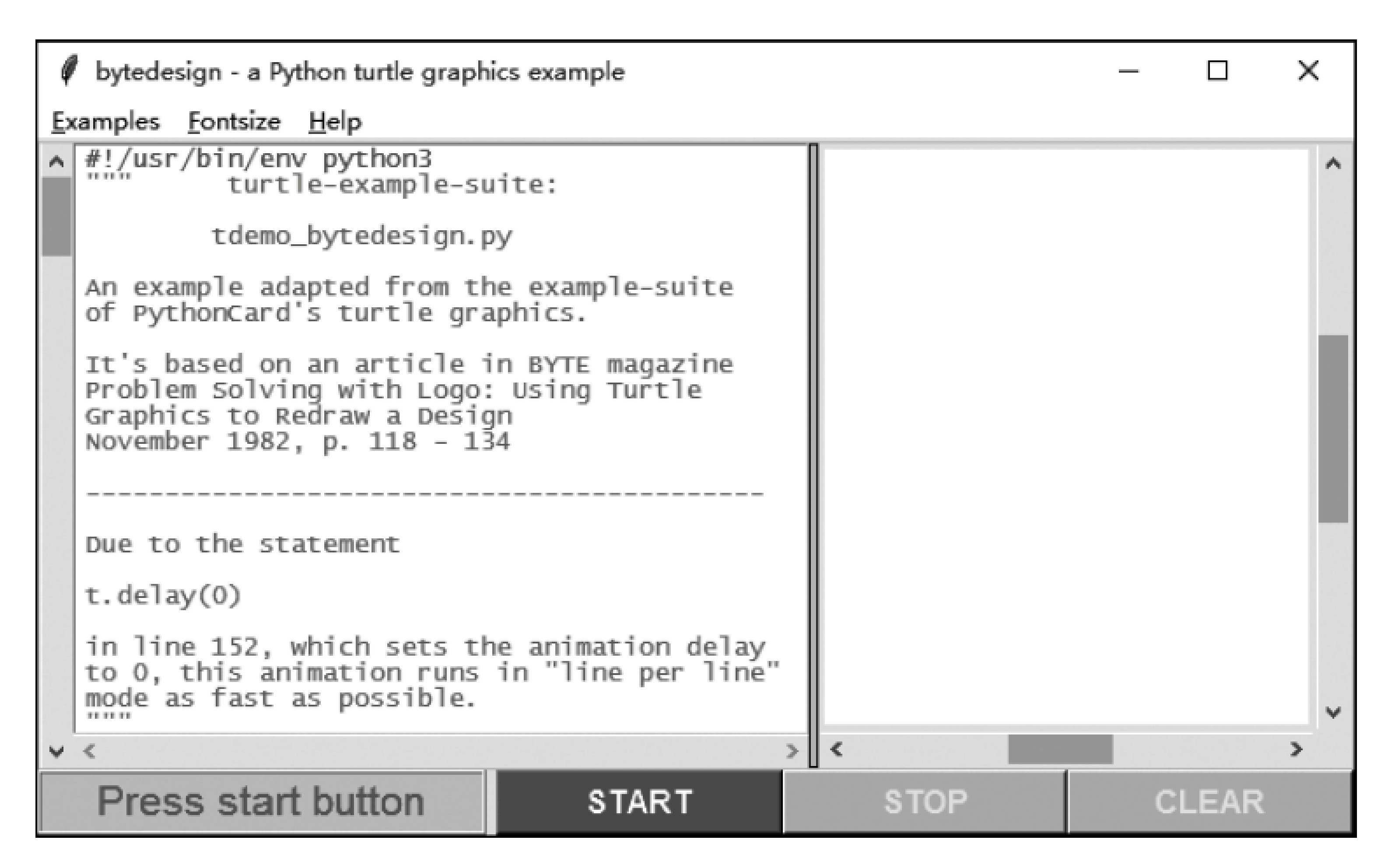


图 11-18　Turtle Demo 演示窗口

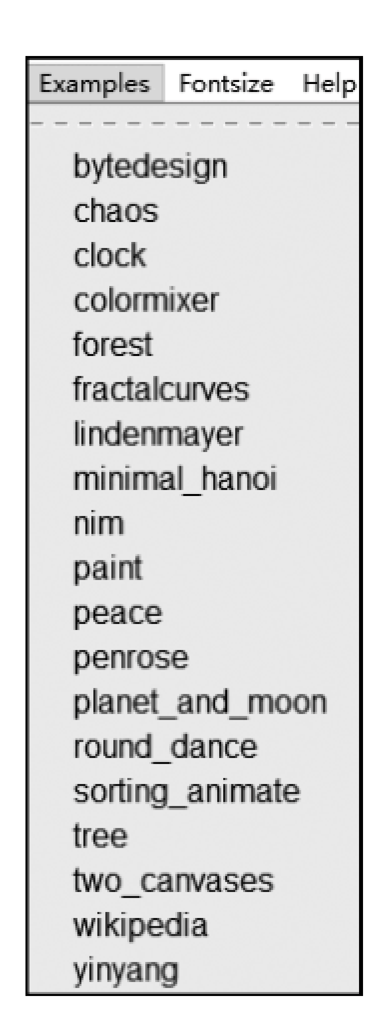


图 11-19　Examples 菜单

例如，选中 Examples|clock 菜单选项，打开如图 11-20 所示的窗口。

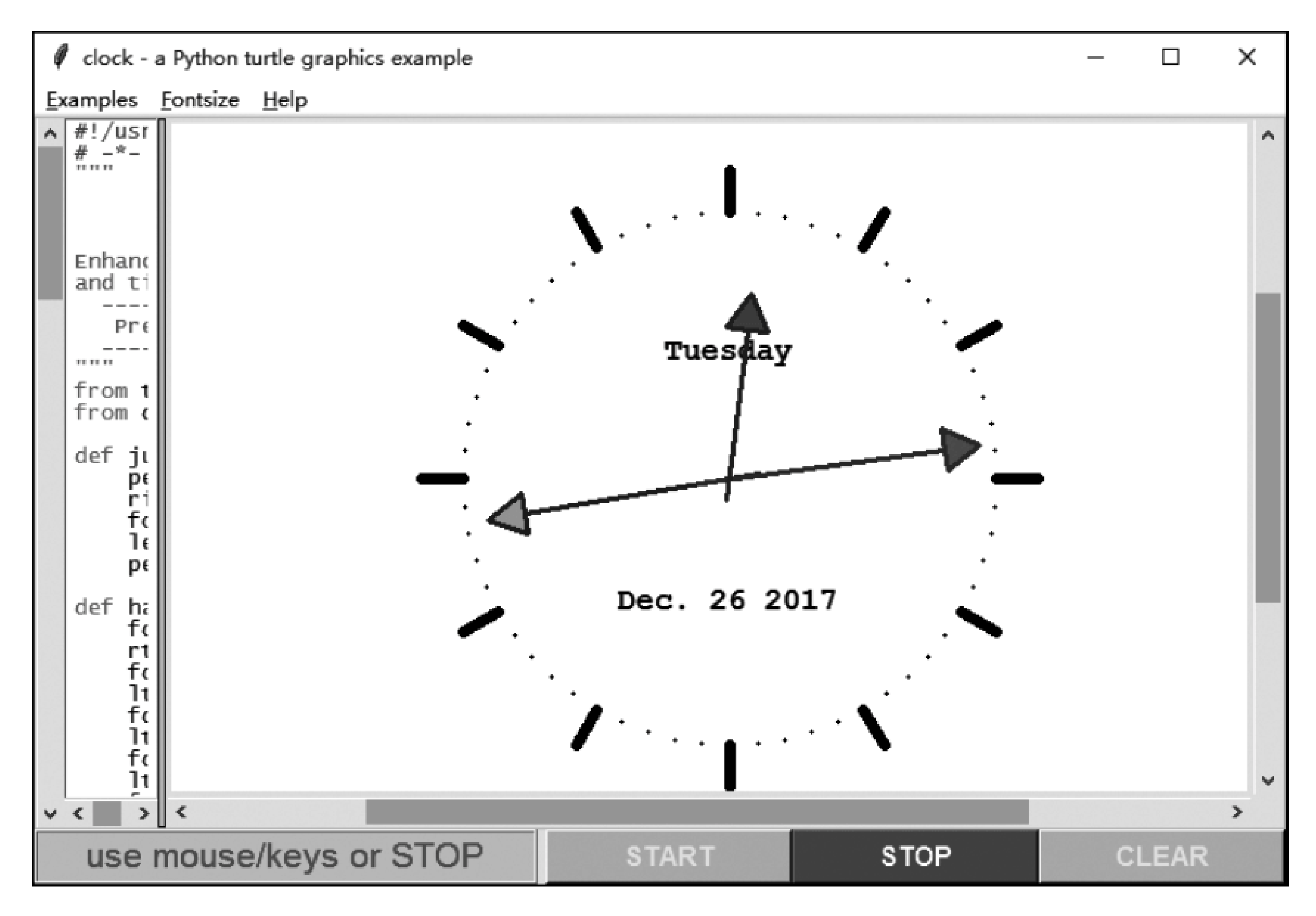


图 11-20　指针式时钟窗口

窗口的左窗格显示的是 clock.py 文件，右窗格是程序运行结果。窗口下方是程序操作说明，右下方设置 3 个操作键，START 键表示运行程序，STOP 键表示暂停程序运行，CLEAR 键表示清除显示。

11.5.2　利用安装文件夹中的演示程序

利用安装文件夹中的文件，也可以体检内置海龟程序。

具体操作过程如下：

在"开始"菜单中选中 IDLE(Python 3.6 64-bit)菜单选项，打开 Python 开发环境，选中

File|Open 菜单选项，如图 11-21 所示，弹出如图 11-22 所示的“打开”对话框。

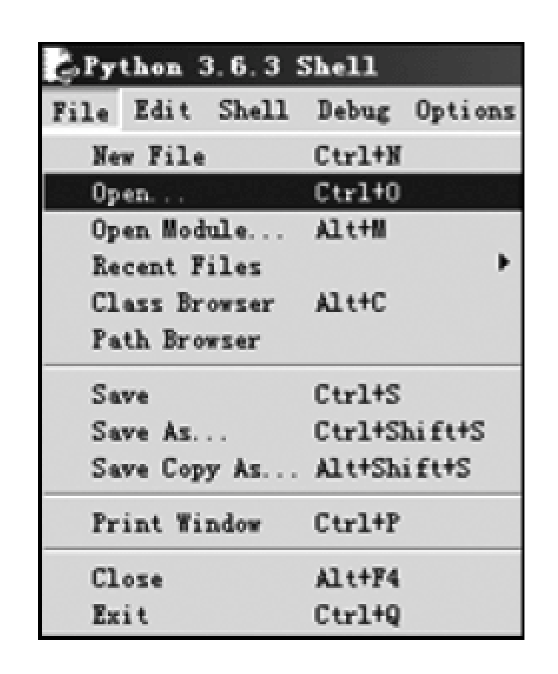


图 11-21　File 菜单

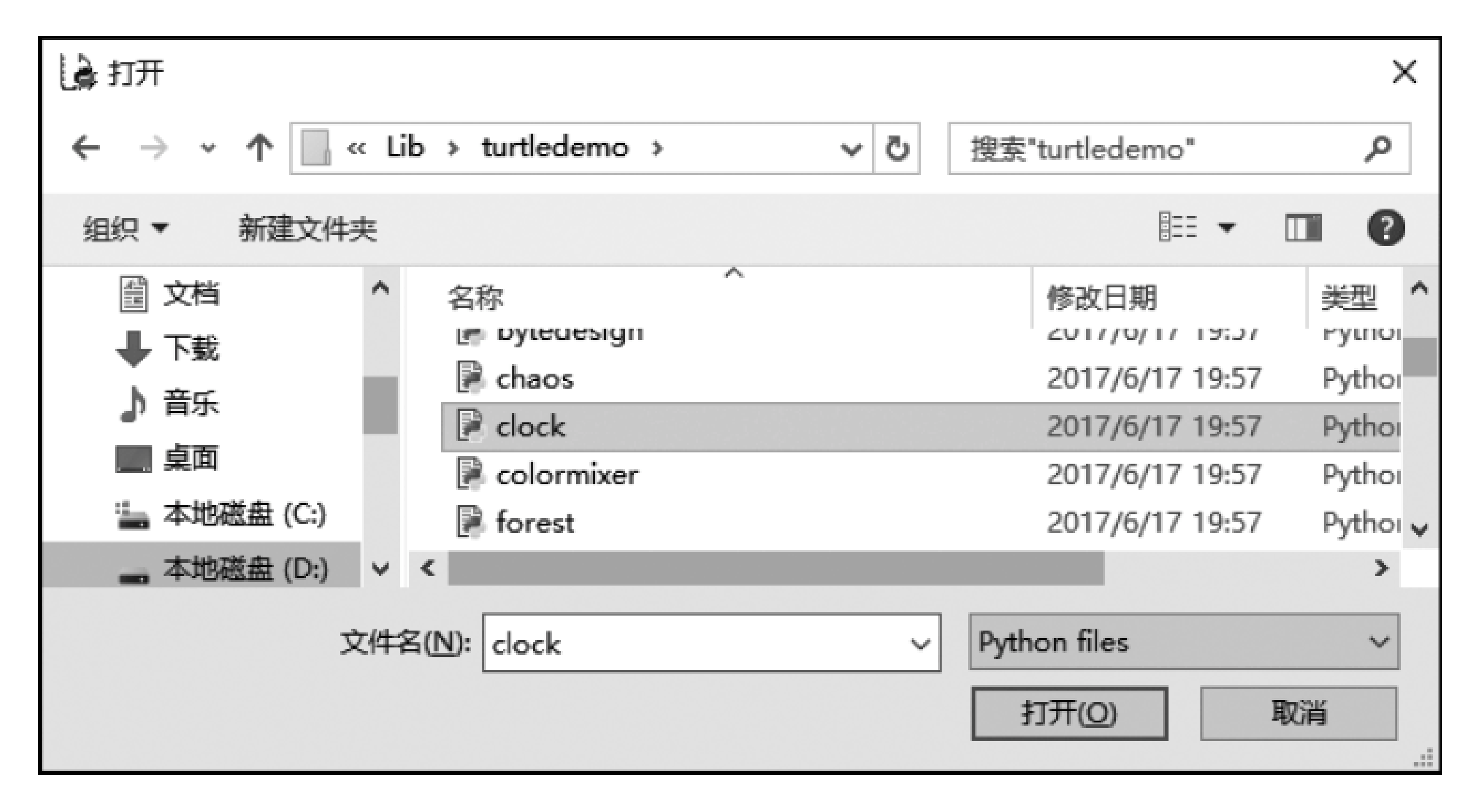


图 11-22　“打开”对话框

展开路径至 D:\Python36\Lib\turtledemo，并选择文件 clock. py 并打开后，系统将弹出内含 Python 程序的编辑窗口，如图 11-23 所示。

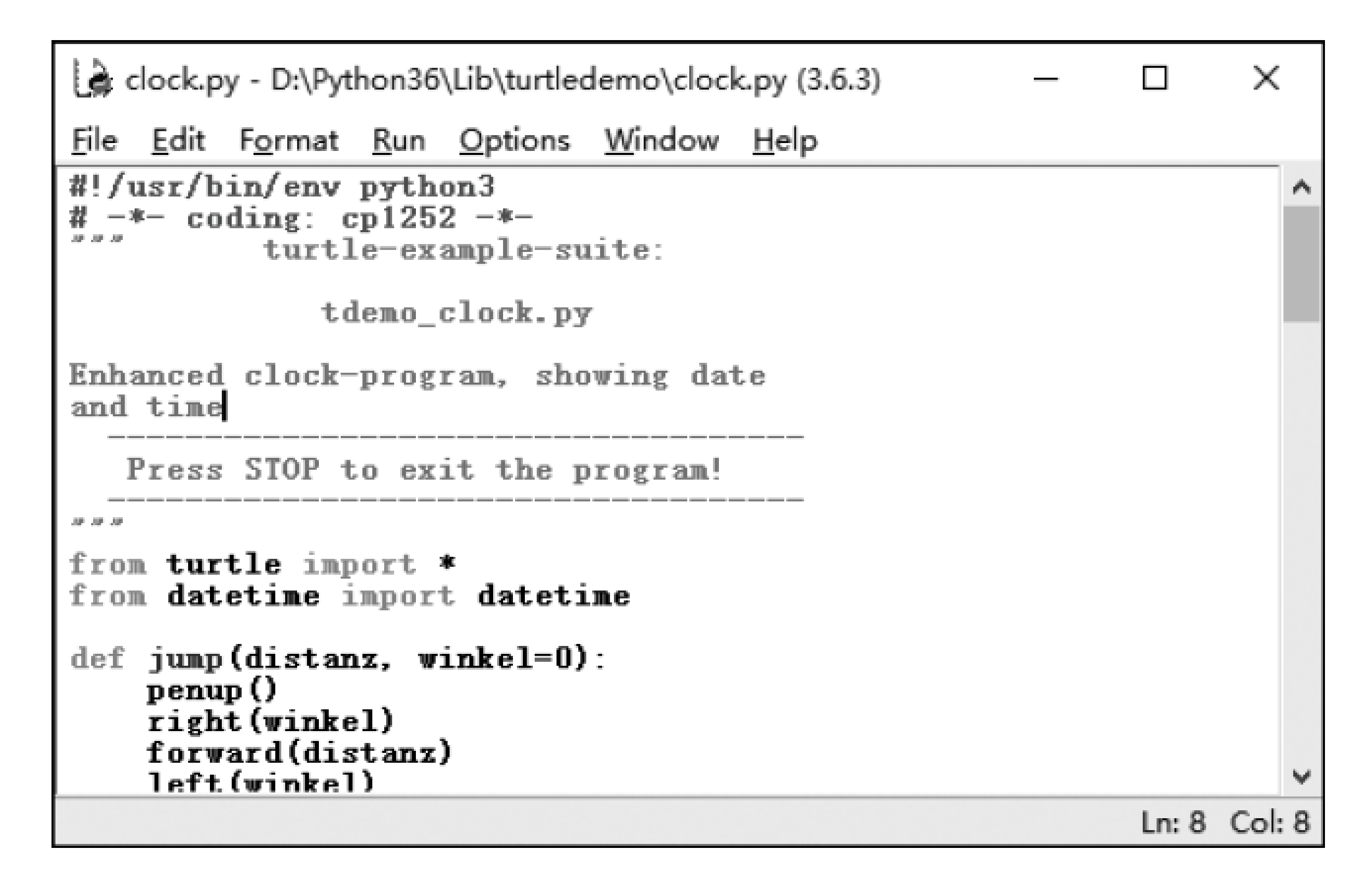


图 11-23　编辑窗口（截图未完）

选中 Run|Run Module 菜单选项或直接按 F5 键，弹出如图 11-24 所示的运行窗口。

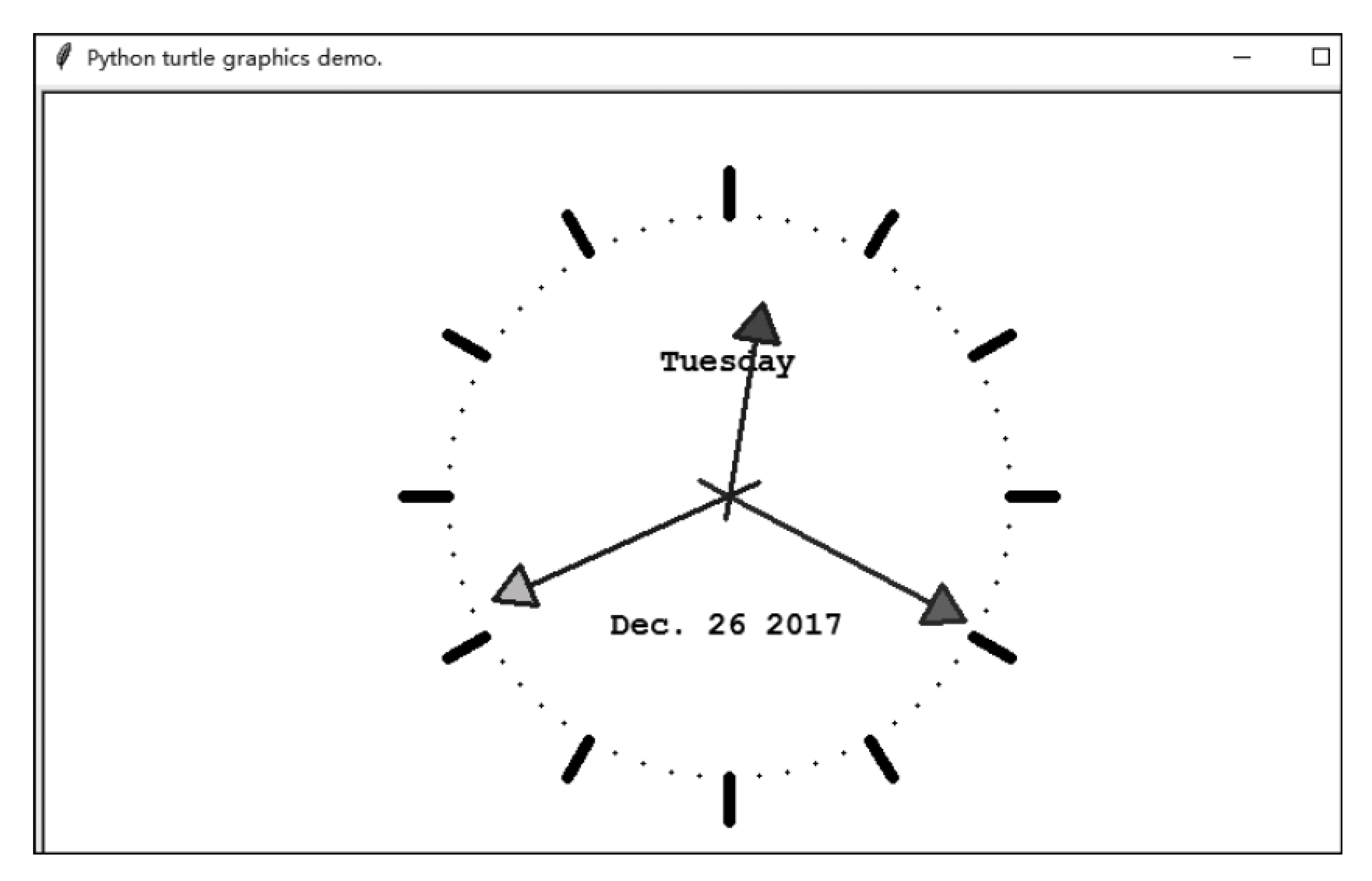


图 11-24　指针式时钟窗口

11.6　turtle 绘图

早在 Python 2.6 版本中就开始引入 turtle(海龟)绘图工具,目前它已经非常完整。

11.6.1　turtle 模块

1. 导入 turtle 模块

在使用 turtle 绘图前,也需要导入 turtle 模块,其一般引用格式如下:

```
from turtle import *
```

在导入 turtle 模块后,程序中才能调用其中的函数和方法。

2. turtle 的属性

turtle 的三大属性如下。

(1) 位置:用于指定 turtle 的初始位置。

(2) 方向:用于指定 turtle 的移动方向。

(3) 画笔:用于设置画笔的属性,例如颜色、线宽等。

3. turtle 的命令

turtle 可以引用许多命令,这些命令通常分为两类,一类是实现运动控制的命令,另一类是控制画笔的命令。

(1) 运动命令。turtle 中的运动命令,如表 11-5 所示。

表 11-5　turtle 的运动命令

命　　令	说　　明
forward(<length>)	向前移动距离<length>代表长度

续表

命　　令	说　　明
backward(＜length＞)	向后移动距离＜length＞代表长度
right(＜degree＞)	向右移动的角度
left(＜degree＞)	向左移动的角度
goto(＜x＞,＜y＞)	将画笔移动到坐标为(x,y)的位置
stamp()	复制当前图形
speed(＜speed＞)	画笔绘制的速度范围[0,10],只能为整数,最快为 10,最慢为 0

(2) 画笔控制命令。turtle 中的画笔控制命令,如表 11-6 所示。

表 11-6　画笔控制命令

命　　令	说　　明
down()	移动时绘制图形,省略时为绘制图形
up()	移动时不绘制图形
pensize(＜width＞)	设置绘图时的宽度
color(＜colorstring＞)	设置绘图时的颜色
fillcolor(＜colorstring＞)	设置绘图的填充颜色

11.6.2　应用案例

【例 11-15】 绘制正方形。

源程序如下:

```
import turtle
import time
#第一次定义画笔颜色
turtle.color("purple")
#定义绘制时画笔的线条宽度
turtle.pensize(5)
#定义绘图速度,由 time 模块控制,此处为较慢的 2
turtle.speed(2)
#以图形坐标(0, 0)为起点进行绘制
turtle.goto(0,0)
#从左上角开始分别以顺时针方向连续绘出正方形的四条边
for i in range(4):
    turtle.forward(100)
    turtle.right(90)
turtle.up()
#设置显示文字的位置,即将画笔移动到(-150, -120)
turtle.goto(-150,-120)
```

```
#第二次定义画笔颜色
turtle.color("red")
#在当前位置显示"Done"信息
turtle.write("Done")
#暂停 3s
time.sleep(3)
```

运行结果如图 11-25 所示。

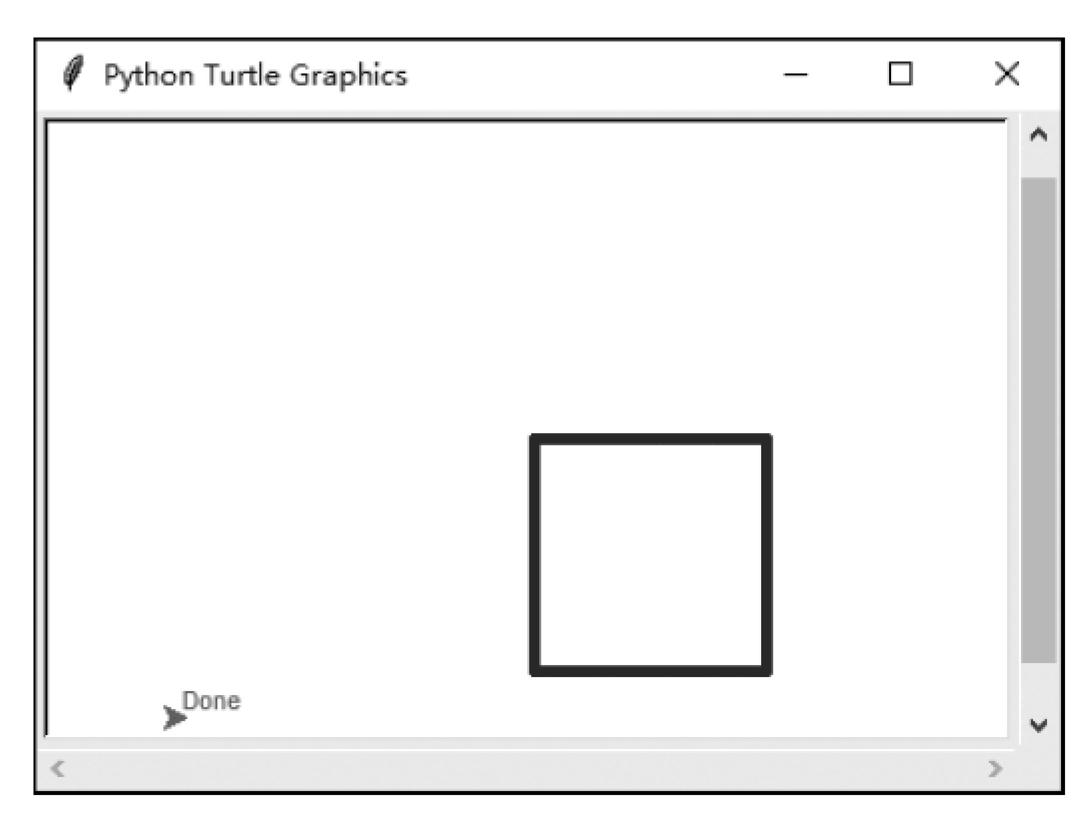


图 11-25　例 11-15 的运行结果

【例 11-16】　绘制五角星。

源程序如下：

```
import turtle
import time
turtle.color("purple")
turtle.pensize(5)
turtle.goto(0,0)
#定义绘图速度，由 time 模块控制，此处为最快的 10
turtle.speed(10)
#从正上方开始分别以顺时针方向连续绘出五角星的 5 条线
for i in range(5):
    #绘制五角星中的 1 条边
    turtle.forward(100)
    #向右旋转 144°
    turtle.right(144)
turtle.up()
turtle.forward(100)
#设置海龟显示文字的位置和颜色
urtle.goto(-150,-120)
turtle.color("red")
turtle.write("Done")
time.sleep(3)
```

运行结果如图 11-26 所示。

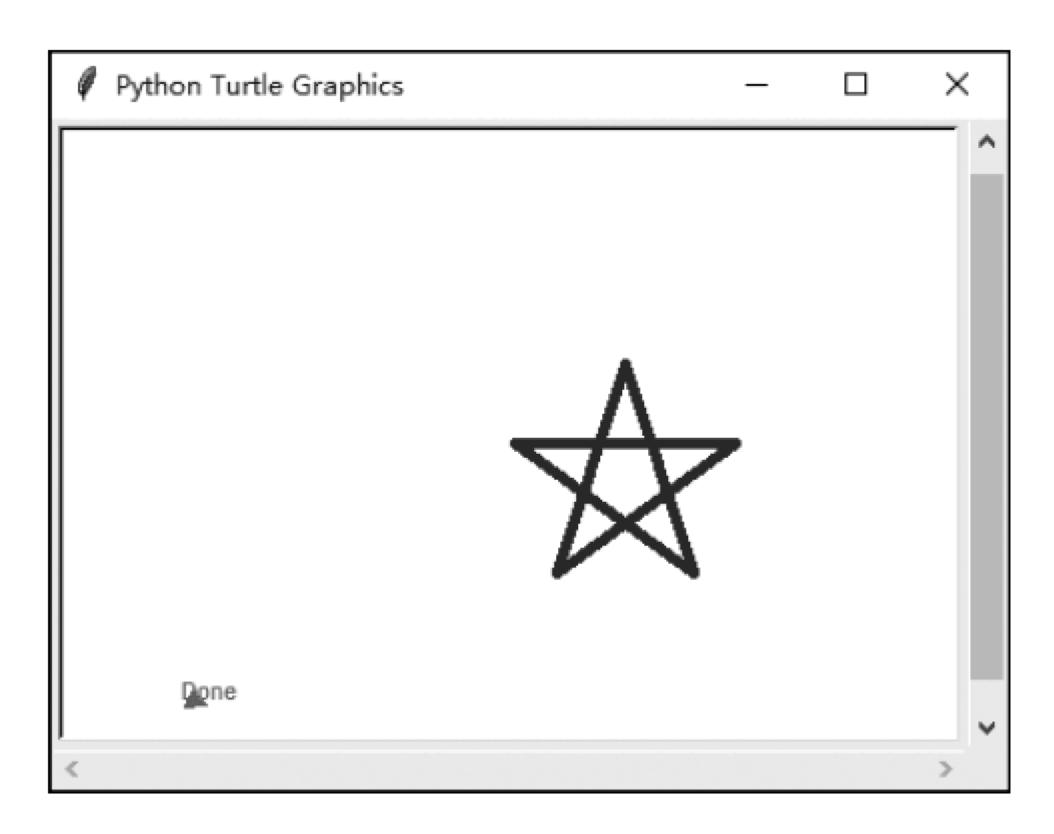


图 11-26　例 11-16 的运行结果

在图 11-26 中的最左侧有一个箭头符号，是显示文字时的起点。

【例 11-17】 绘制角度不同的重复线。

源程序如下：

```
from turtle import *
#定义画笔颜色
color("blue")
#从最左侧开始分别以反时针方向连续绘出 9 条线
while True:
    #绘制一条线
    forward(240)
    #向左旋转 160 度
    left(160)
    #结束绘制直线
    if abs(pos()) <1 : break
print("Over!")
```

本例中的循环条件为 True，但在循环体中绘制第 9 条线时将与最左侧坐标点重合，这时函数 abs(pos())值小于 1 表示退出循环。

运行结果如图 11-27 所示。

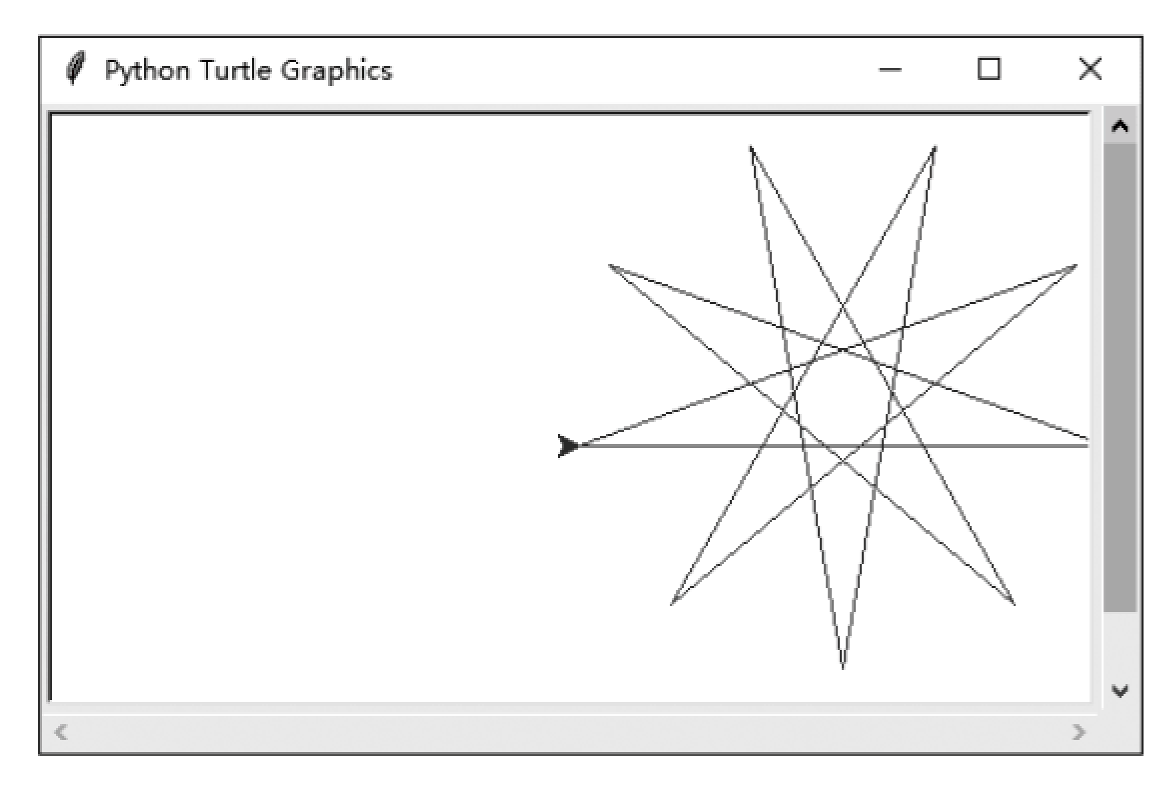


图 11-27　例 11-17 的运行结果

在图 11-27 中的最左侧有一个向右的箭头符号，是绘图时的起点。

【例 11-18】 绘制一组正 n 边形（$1 \leqslant n \leqslant 8$）。

源程序如下：

```
import turtle
#绘制指定边长的正多边形
def polygons(sides,side_len):
    for i in range(sides):
        #绘制正多边形的一条边
        turtle.forward(side_len)
        #旋转角度并准备绘制下一条边
        turtle.left(360.0/sides)
#绘制三角形、正方形、正五边形、…、正八边形
for i in range(3,9):
    #设置初始边长为 80 个像素
    step=80
    #绘制指定长度的多边形
    polygons(i,step)
```

运行结果如图 11-28 所示。

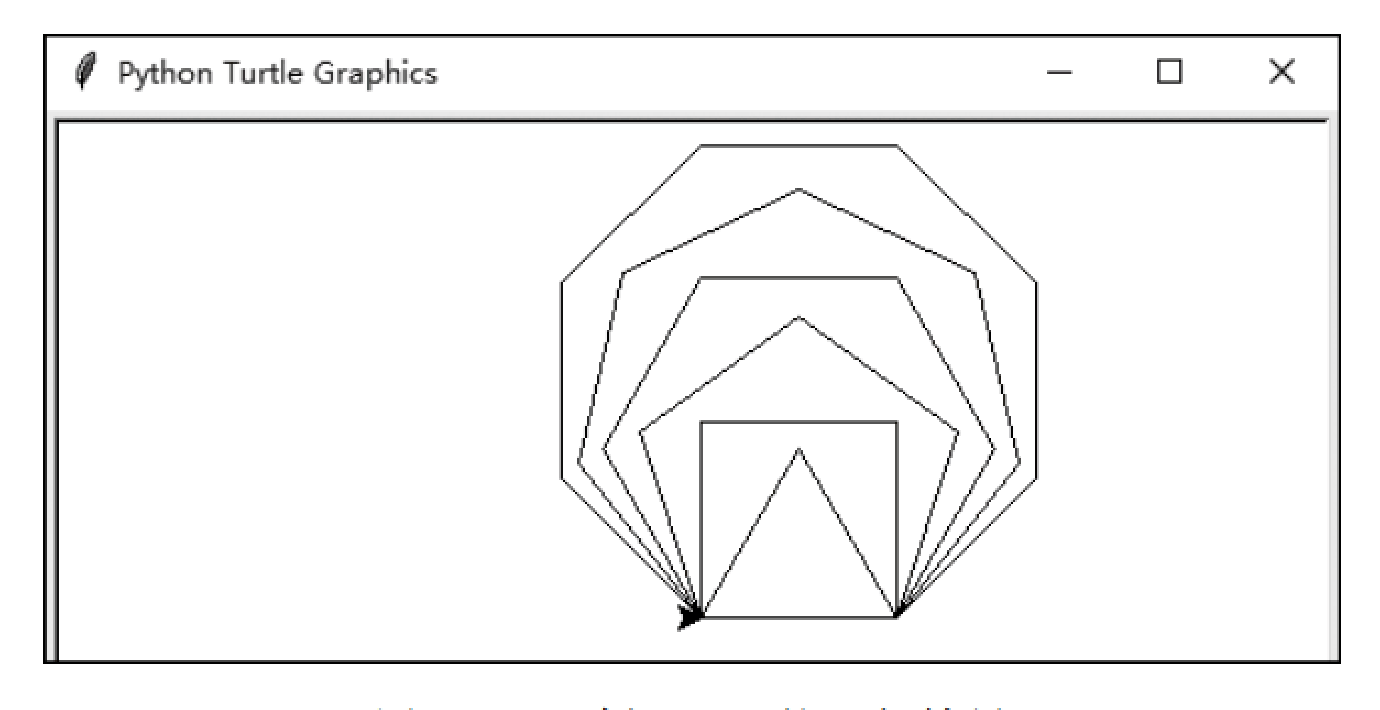


图 11-28　例 11-18 的运行结果

【例 11-19】 绘制 4 个不同颜色且嵌套的螺线。

源程序如下：

```
import turtle
t=turtle.Turtle()
#定义绘图速度，由 time 模块控制，此处为最慢的 0
t.speed(0)
#设置颜色元组为红、蓝、绿、黄
colors=["red","blue","green","yellow"]
for i in range(0,60,1):
    #选择画笔颜色
    t.pencolor(colors[i%4])
    #绘制圆
    t.circle(i)
```

```
    #向左旋转 91°
    t.left(91)
print("Over!")                    #在 IOLE 环境中显示 Over!
```

运行结果如图 11-29 所示。

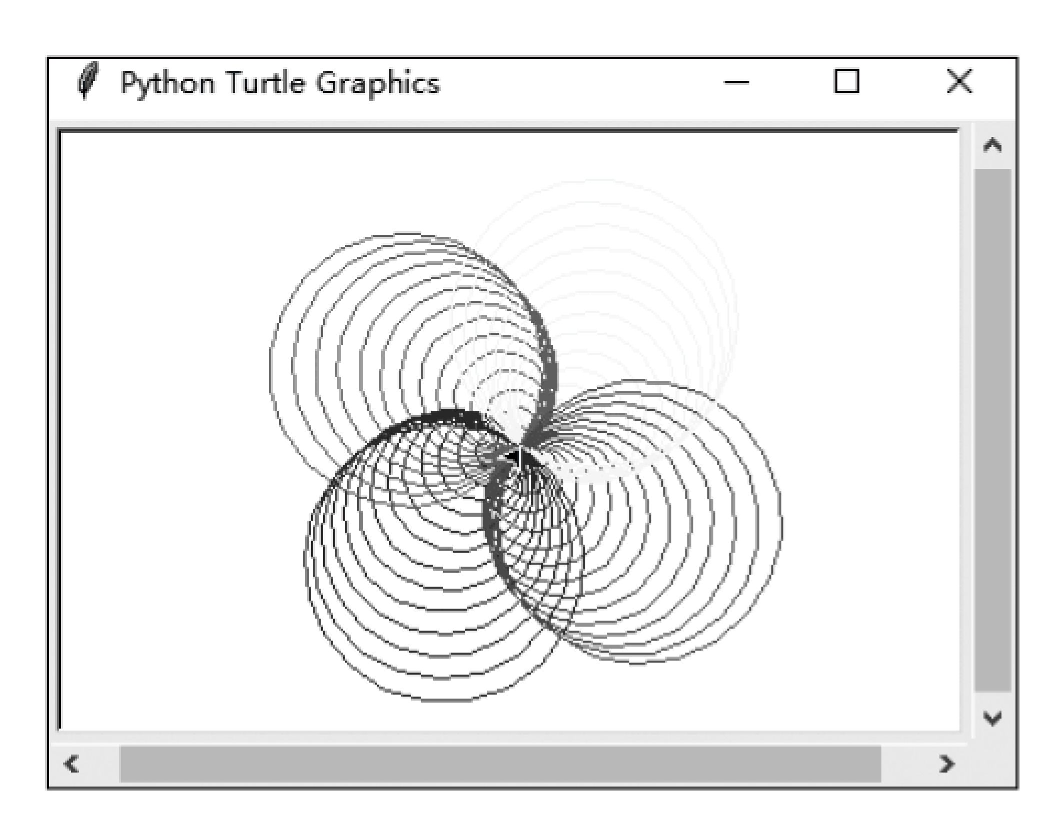


图 11-29　例 11-19 的运行结果

在图 11-29 中,不同颜色的螺线分别为 15 条,共 60 条,由第 7 行的 for 循环生成。

11.7　分 形 图 形

分形是以数学方法来模拟自然存在的、科学研究中出现树叶、云雾等各种非规则图形,从而产生分形艺术图形。这是一个非常复杂的领域,本节只介绍如何利用 turtle 模块绘制 Koch 曲线、Hilbert 曲线和分形树。

11.7.1　Koch 曲线

【例 11-20】　绘制 Koch 曲线。Koch 曲线的大意是,从一条直线段开始,将线段中间的三分之一部分用一个等边三角形的两边代替。在新的图形中,又将图中每一直线段中间的三分之一部分都用一个等边三角形的两条边代替,再次形成新的图形……如此迭代,最后形成 Koch 曲线,如图 11-30 所示。

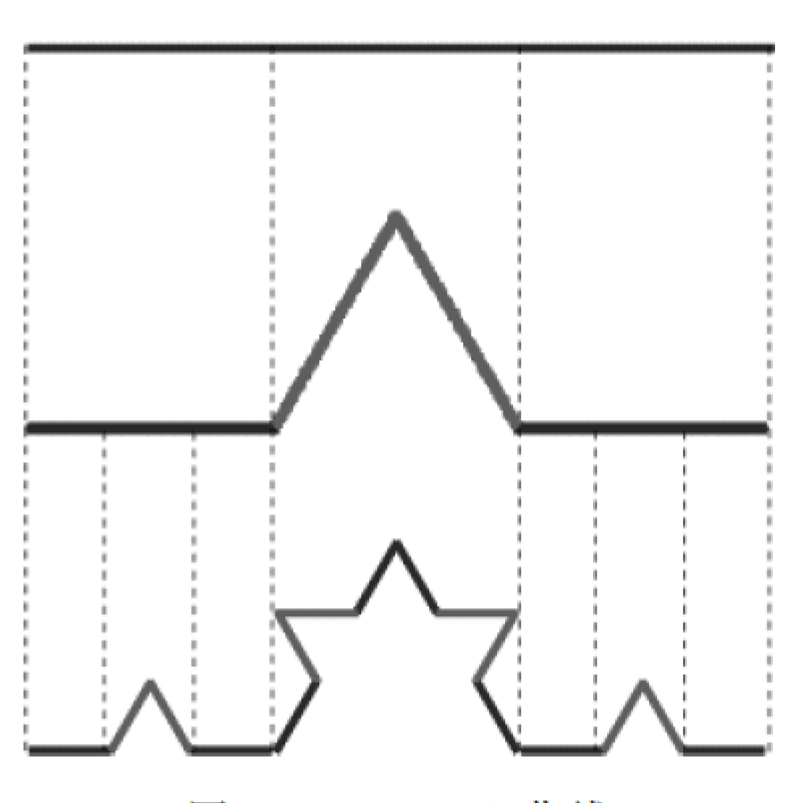


图 11-30　Koch 曲线

求解方法:要生成 Koch 曲线最好使用递归函数。在递归降阶过程中,直线长度按等比缩小为 1/3,旋转角度以反时针为基准,即首先反时针旋转 60°绘制第一条直线,然后顺时针旋转 120°绘制第二条直线,最后再反时针旋转 60°绘制第 3 条直线。

源程序如下:

```
import sys
import turtle
```

```
#定义递归函数
def koch(t,order,length):
    if order==0:
        #绘制一条直线, 表示 0 阶 koch 曲线
        t.forward(length)
    else:
        #递归函数降阶, 并将直线长度等比缩小为 1/3
        koch(t,order-1,length/3)
        #向左旋转 60°并控制第 1 条直线
        t.left(60)
        koch(t,order-1,length/3)
        #向右旋转 120°并控制第 2 条直线
        t.left(-120)
        koch(t,order-1,length/3)
        #向左旋转 60°并控制第 3 条直线
        t.left(60)
        koch(t,order-1,length/3)
#调用二阶 koch 曲线
n=2
#设置 koch 曲线的最长边为 400 像素
least=400
kch=turtle.Turtle()
#调用递归函数
koch(kch,n,least)
```

运行结果如图 11-31 所示。

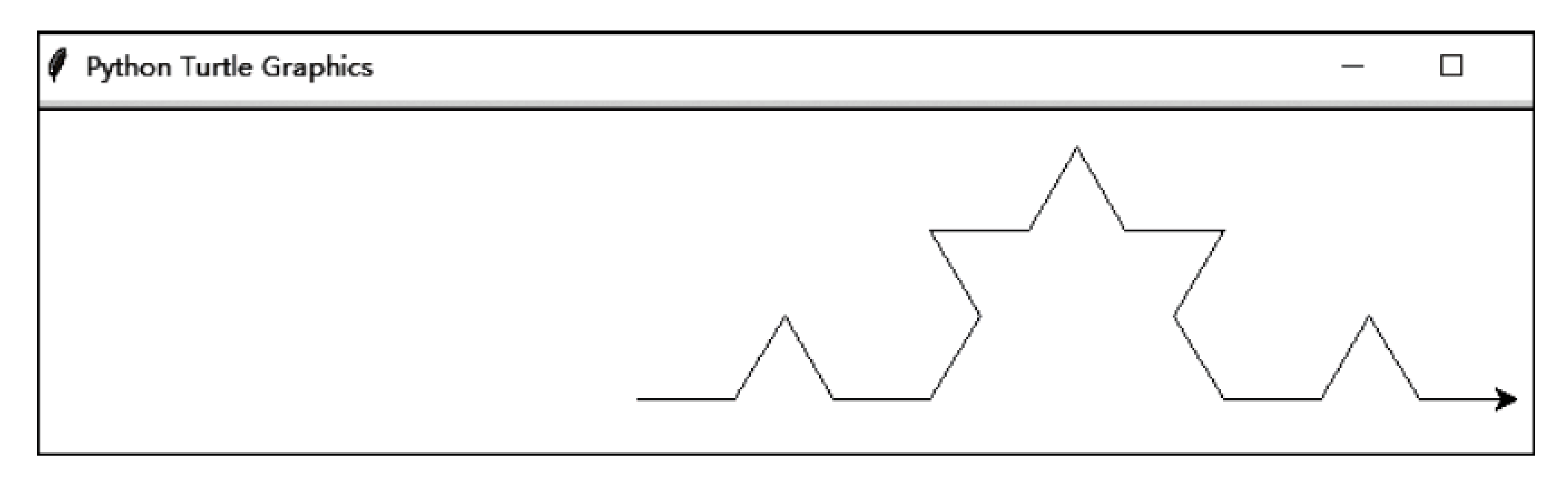


图 11-31　例 11-20 的运行结果

11.7.2　Hilbert 曲线

【例 11-21】 绘制 Hilbert 曲线。Hilbert 曲线的大意是,取一个正方形并且把它分出 9 个相等的小正方形,然后从左下角的正方形开始至右上角的正方形结束,依次把小正方形的中心用线段连接起来;下一步把每个小正方形分成 9 个相等的正方形,然后从上述方式把小正方形的中心连接起来……将这种操作过程继续进行下去,最终得到 Hilbert 曲线,如图 11-32 所示。

求解方法:要生成 Hilbert 曲线最好使用递归函数,方法与生成 Koch 曲线类似。

源程序如下:

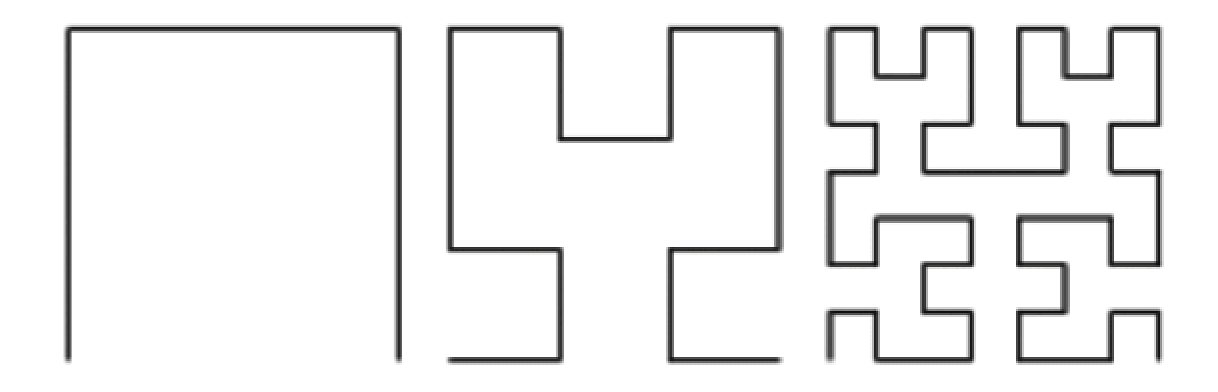


图 11-32　Hilbert 曲线

```
from turtle import *
#定义类 CurvesTurtle
class CurvesTurtle(Pen):
    #定义递归函数 hilbert()
    def hilbert(self,size,level,parity):
        if level==0:
            #表示 0 阶 hilbert 曲线
            return
        self.left(parity*90)
        #递归函数降阶，并绘制第 1 个 hilbert 子曲线
        self.hilbert(size,level-1,-parity)
        #绘制第 2 个 hilbert 子曲线
        self.forward(size)
        self.right(parity*90)
        self.hilbert(size,level-1,parity)
        #绘制第 3 个 hilbert 子曲线
        self.forward(size)
        self.hilbert(size,level-1,parity)
        #绘制第 4 个 hilbert 子曲线
        self.right(parity*90)
        self.forward(size)
        self.hilbert(size,level-1,-parity)
        #旋转至下次递归操作位置
        self.left(parity*90)
#定义函数实现递归调用
def main():
    ft=CurvesTurtle()
    ft.reset()
    ft.ht()
    ft.getscreen().tracer(1,0)
    ft.pu()
    size=6
    ft.setpos(-33*size,-32*size)
    ft.pd()
    ft.hilbert(size,6,1)
    #创建显示框架
    ft.fd(size)
```

```
    for i in range(3):
        ft.lt(90)
        ft.fd(size*(64+i%2))
    ft.pu()
    for i in range(2):
        ft.fd(size)
        ft.rt(90)
    ft.pd()
    for i in range(3):
        ft.fd(size*(66+i%2))
        ft.rt(90)
    ft.end_fill()
#测试代码
if __name__=="__main__":
    msg=main()
    print(msg)
    mainloop()
```

运行结果如图 11-33 所示。

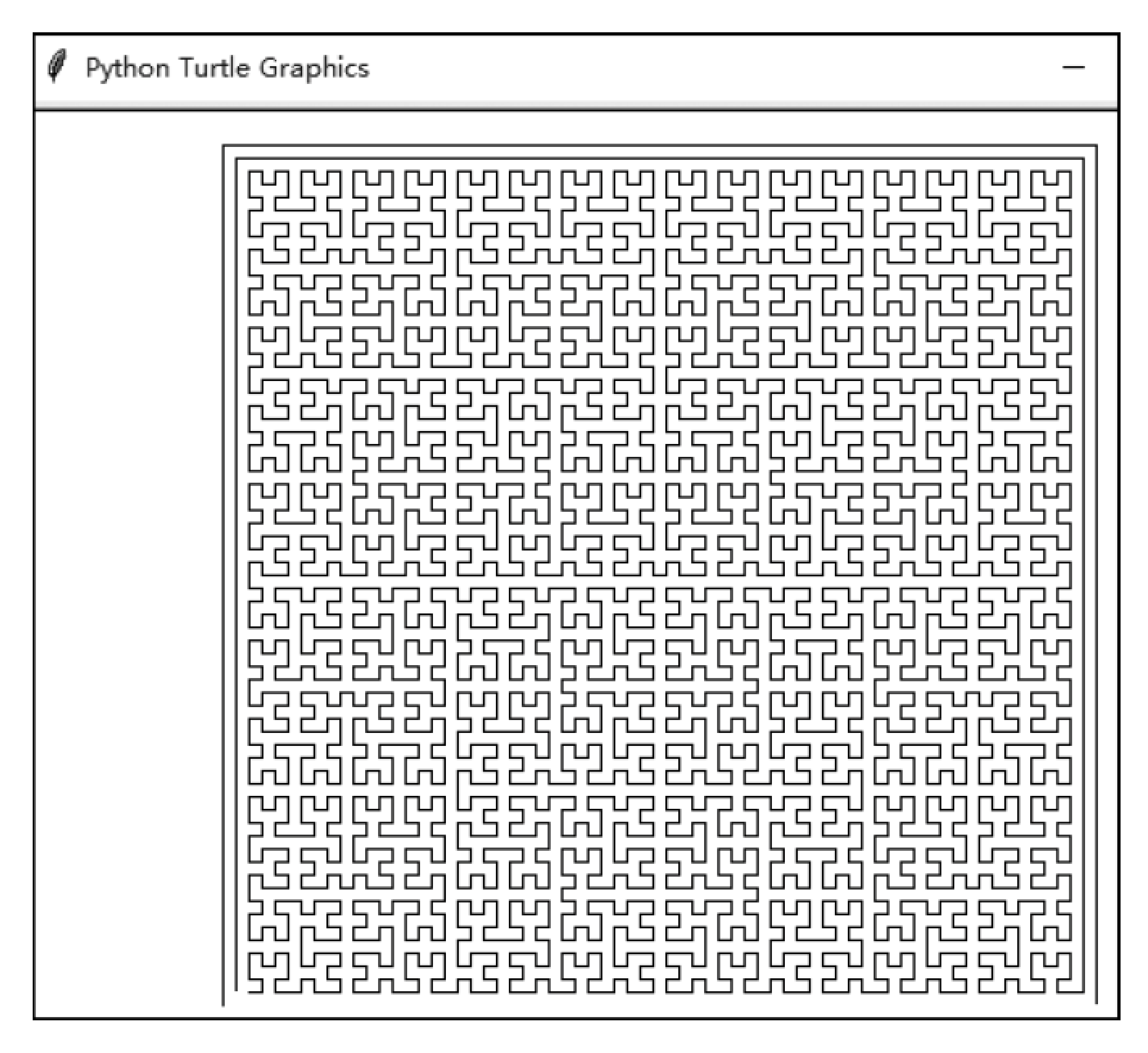


图 11-33　例 11-21 的运行结果

11.7.3　分形树

【例 11-22】　绘制分形树，如图 11-34 所示。

求解方法：要生成分形树最好使用递归函数，方法与生成 Koch 曲线类似。

源程序如下：

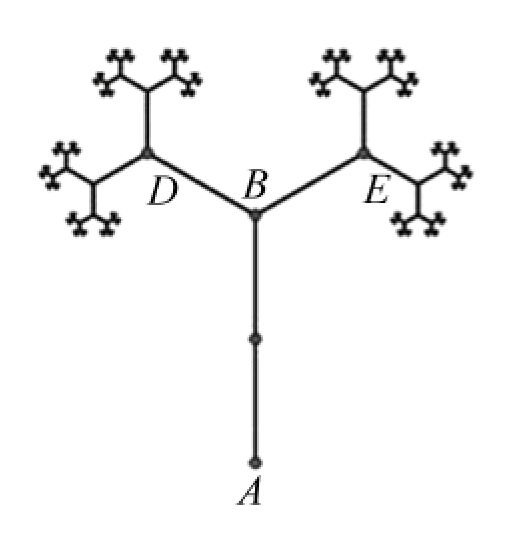


图 11-34　原始分形树

```
from turtle import Turtle,mainloop
#定义函数,其中参数 l, a, f 分别表示分枝的长度、角度折半和缩短比例
def tree(plist,l,a,f):
    if l>4:
        lst=[]
        for p in plist:
            p.forward(l)
            #在每个分枝点上递归调用当前画笔(共 51 个)
            q=p.clone()
            p.left(a)
            q.right(a)
            lst.append(p)
            lst.append(q)
        #递归调用
        for x in tree(lst,l*f,a,f) : yield None
#定义函数绘制分形树
def maketree():
    p=Turtle()
    p.setundobuffer(None)
    p.hideturtle()
    p.getscreen().tracer(30,0)
    p.left(90)
    p.penup()
    p.forward(-210)
    p.pendown()
    #开始递归调用
    t=tree([p],200,65,0.6375)
    for x in t : pass
    print(len(p.getscreen().turtles()))
#定义函数实现递归调用
def main():
    maketree()
#测试代码
if __name__=="__main__":
    msg=main()
    print(msg)
```

```
    mainloop()
```

运行结果如图 11-35 所示。

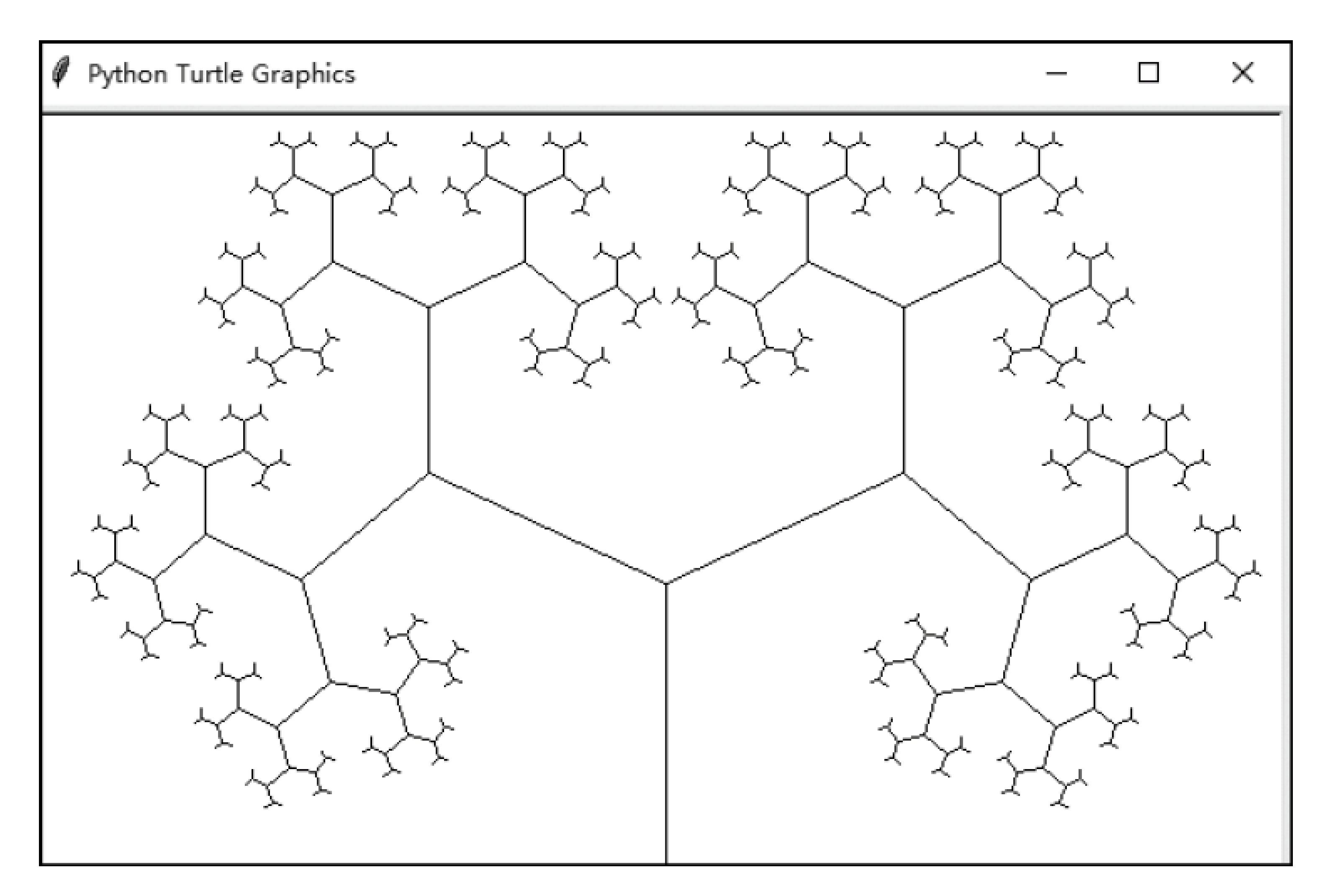


图 11-35　例 11-22 的运行结果

11.8　显示字体

下面介绍显示字体的两种方法，即通过元组显示字体和通过 Font 对象显示字体。

11.8.1　通过元组显示字体

通过一个含 3 个元素的元组可以显示字体，其一般格式如下：

```
(<font family>, <size>, <modifiers>)
```

说明：

(1) <font family>显示字体的名称；

(2) <size>表示以像素为单位的字体大小；

(3) <modifiers>表示粗体、斜体或下画线的字型。

【例 11-23】 通过元组显示字体。

源程序如下：

```
from tkinter import *
root=Tk()
root.title("通过元组显示字体")
#创建含 7 种字体的元组
fnt=("Arial",("MS Serif",),"Symbol","System",("Times New Roman",),"Fixdsys",
     "Verdana")
for ft in fnt:
    #创建 Label 组件并指定字体显示文本
    Label(root,text="Representing font by tuple",font=ft).grid()
```

```
root.mainloop()
```

运行结果如图 11-36 所示。

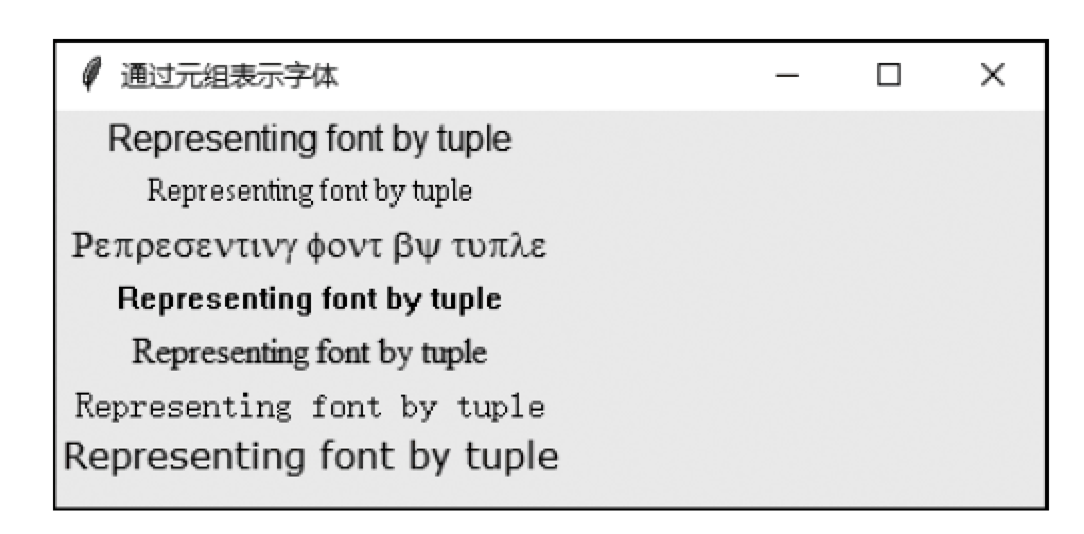


图 11-36　例 11-23 的运行结果

11.8.2　通过 Font 对象显示字体

显示字体的另一种方式是使用 tkFont. Font 来创建字体，其一般格式如下：

```
tkFont.Font(<family>="字体名", <size>, <weight>, <slant>, <underline>, <overstrike>)
```

这里的<size>显示字体大小；<weight>指定粗体(bold)或正文(normal)；<slant>指定斜体(italic)或正文(normal)；<underline>值为 1 表示下画线；<overstrike>值为 1 表示删除线。

例如，创建字体对象：

```
ft =Font((amily="Helvetica", size=36, weight="bold")
```

显示字体对象 ft 的属性为 Helvetica 字体，36 号，粗体。

【例 11-24】　通过 Font 对象显示字体。

源程序如下：

```
from tkinter import *
#导入字体模块
import tkinter.font
root=Tk()
root.title("通过 Font 对象显示字体")
#按指定字体样式创建第一个 Font 对象
ft1=tkinter.font.Font(family="Times New Roman",size=18,weight="bold")
#第一次创建 Label 控件并显示文本
Label(root,text="Representing fonts through Font objects",font=ft1).grid()
#按指定字体样式创建第二个 Font 对象
ft2=tkinter.font.Font(family="Times New Roman",size=16,weight="bold")
#第二次创建 Label 控件并显示文本
Label(root,text="Python is an easy to learn, powerful programming language",font=
ft2).grid()
root.mainloop()
```

运行结果如图 11-37 所示。

通过Font对象表示字体

Representing fonts through Font objects

Python is an easy to learn, powerful programming language

图 11-37　例 11-24 的运行结果

习　题　11

一、简答题

1. 什么是 Canvas(画布)？ 它与 Tkinter 模块是什么关系？
2. 如何创建 Canvas 组件？
3. 简述屏幕坐标的设置模式。
4. Canvas 含有哪些图形绘制函数？
5. Canvas 含有哪些控制图形函数？
6. 简述 turtle 模块的主要功能。
7. 如何调用 IDLE 环境中的 Turtle Demo 演示程序？
8. turtle 有哪些基本属性？
9. turtle 有哪些实现运动控制的命令？
10. turtle 有哪些控制画笔的命令？
11. 简述分形图形的主要含义。
12. 如何通过元组显示字体？
13. 如何通过 Font 对象显示字体？

二、编程题

1. 由 4 条直线分别绘制正方形、矩形和平行四边形。
2. 由 3 条直线分别绘制直角三角形、等腰三角形和等边三角形。
3. 调用 create_polygon()函数分别绘制正方形、矩形和平行四边形。
4. 调用 create_polygon()函数分别绘制直角三角形、等腰三角形和等边三角形。
5. 调用 create_rectangle()函数绘制 4 个正方形，并能够分别构成 L 形和 T 形图案。
6. 调用 create_arc()函数分两行绘制 3 个平行的相邻圆和 3 个平行的相邻椭圆。
7. 调用 create_image()函数显示 3 个平行的图像。
8. 调用 create_text()函数显示如下 3 行的信息。

```
******************************
*          计算思维          *
******************************
```

9. 调用 create_rectangle()函数绘制一个矩形，并调用 move()函数将其移动。
10. 调用 create_rectangle()函数绘制一个正方形，并调用 scale()函数将其缩放。
11. 绘制余弦函数曲线。
12. 调用 IDLE 环境中的 Turtle Demo 选项，显示一个海龟图形。

13. 调用安装文件夹中的示例程序，显示一个海龟图形。

14. 调用 turtle 模块绘制完全嵌套的两个正方形。

15. 调用 turtle 模块绘制完全嵌套的两个矩形。

16. 在字体元组中设定 3 种字体，并分别显示字符串"Define the font from the font tuple"。

17. 在 Font 对象中设定一种字体，并分别显示 3 种字号的字符串"Defining fonts from Font objects"。

第二部分　实　　验

本部分共安排 7 个实验，内容涉及程序设计与计算思维方面的机器实现。主要包括数据与计算、流程控制、函数、数据文件、面向对象编程、图形界面设计和绘制曲线。

实验Ⅰ　数据与计算

1. 实验目标

（1）理解并掌握计算环境中的计算思维概念，例如抽象、自动化、有限性和数据表示。

（2）理解并掌握 Python 编码风格。

（3）理解并掌握组合符号、标识符、关键字和预定义标识符的概念与运用。

（4）理解并掌握基本数据类型与复合数据类型的概念与运用。

（5）理解并掌握不可变数据类型与可变数据类型概念与运用。

（6）理解并掌握常量和变量的概念与运用。

（7）理解并掌握 4 种数字类型的概念与运用。

（8）理解并掌握字符串的概念与运用。

（9）理解并掌握序列的概念与运用。

（10）理解并掌握列表的概念与运用。

2. 实验环境

（1）Windows 7 操作系统及其以上版本。

（2）Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅰ-1】 整型数据的运算。

在 IDLE 编辑窗口中输入如下程序，并命名为 p12_A1.py。

```
a=1*2*3*4*5*6*7*8*9*10*11*12*13*14*15*16*17*18*19*20*21
b=1*2*3*4*5*6*7*8*9*10*11*12*13*14*15*16*17*18*19*20*21*22
print("a=",a)
print("b=",b)
```

程序调试完成后，填写运行结果：

【实验Ⅰ-2】 浮点型数据的运算。

在 IDLE 编辑窗口中输入如下程序，并命名为 p12_A2.py。

```
a=1234567890000000000
b=a+20
print("a=",a)
print("b=",b)
```

程序调试完成后，填写运行结果：

【实验Ⅰ-3】 生成范围为 0～10000 的 3 个随机整数。

在 IDLE 编辑窗口中,输入如下命令:

```
>>>import random
>>>print(random.randint(0,1001))
>>>print(random.randint(0,1001))
>>>print(random.randint(0,1001))
```

程序调试完成后,填写运行结果:

第 1 个随机数	第 2 个随机数	第 3 个随机数

【实验Ⅰ-4】 把列表当作队列使用。

队列(Queue)是一种采用先进先出(First In First Out,FIFO)策略的数据结构,就是现实生活中的排队策略。例如,顾客在结账时,按照队列的先后顺序结账,即先来的顾客先结账,后来的顾客后结账。列表可以实现队列操作,出队操作是删除列表中的第一个元素,入队操作是在列表的最后添加一个元素。

在 IDLE 交互环境中,通过操作实现如下要求:

(1) 初始化列表:

```
["monday", "tuesday", "thursday", "wednesday", "friday"]
```

在 IDLE 编辑窗口中,输入如下命令:

```
>>>my_list=["monday","tuesday","thursday","wednesday","friday"]
```

(2) 入队操作结果:

```
["monday", "tuesday", "thursday", "wednesday", "friday", "saturday", "sunday"]
```

在 IDLE 编辑窗口中,输入如下命令:

```
>>>my_list=["monday","tuesday","thursday","wednesday","friday"]
>>>my_list.append("saturday")
>>>my_list.append("sunday")
>>>my_list
```

(3) 出队操作结果:

```
["thursday", "wednesday", "friday", "saturday", "sunday"]
```

在 IDLE 编辑窗口中,输入如下命令:

```
>>>my_list=["monday", "tuesday", "thursday", "wednesday", "friday", "saturday", "sunday"]
```

```
>>>del my_list[0]
>>>del my_list[0]
```

操作正确后要求将操作过程与运行结果截图,并存入“实验 1.doc”文档中。

【实验Ⅰ-5】 编程显示如下形式的直角三角形。

```
*
**
***
****
*****
******
*******
```

在 IDLE 编辑窗口中输入程序,并命名为 p12_A3.py。

【实验Ⅰ-6】 分别初始化两个长度为 20 的空格串和星号串,使用切片操作输出如下形式的平行四边形。

```
********************
 ********************
  ********************
   ********************
    ********************
     ********************
      ********************
```

在 IDLE 编辑窗口中输入程序,并命名为 p12_A4.py。

【实验Ⅰ-7】 使用如下 3 个公式计算圆周率:

(1) 3+1/7;

(2) (3+1/(7+1/15));

(3) (3+1/(7+1/(15+1/25)))。

要求浮点数的小数部分为 6 位精度。

在 IDLE 编辑窗口中输入程序,并命名为 p12_A5.py。

【实验Ⅰ-8】 从键盘输入直角三角形的两条直角边长,计算周长和面积。

在 IDLE 编辑窗口中输入程序,并命名为 p12_A6.py。

在完成实验后,将文档“实验 1.doc”和 6 个源程序文件发送到教师邮箱中。

实验Ⅱ　流程控制

1. 实验目标

(1) 理解并掌握编程技术中的计算思维：穷举、迭代、分治、模块和约简。

(2) 掌握并实现分支选择。

(3) 掌握并实现 range()函数。

(4) 掌握并实现循环程序。

(5) 掌握并实现循环嵌套。

(6) 掌握并实现 continue、break 和 pass 语句的运用。

(7) 掌握并实现一维列表处理。

(8) 掌握并实现二维列表处理。

(9) 掌握并实现字符串处理。

(10) 掌握并实现查找与排序程序。

2. 实验环境

(1) Windows 7 操作系统及其以上版本。

(2) Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅱ-1】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_B1.py。

```
a=[1,2,2,3,4,3,4,5,1,5,1]
n=0
for i in range(0,10-n,1):
    num=a[i]
    for j in range(i+1,10-n,1):
        if a[j]==num:
            for k in range(j,10-n,1):
                a[k]=a[k+1]
                n=n+1
for i in range(0,10-n,1) : print(a[i],end="\t")
```

程序调试完成后，填写运行结果：

【实验Ⅱ-2】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_B2.py。

```
b=[[1,2,3,4],[5,6,7,8]]
for i in range(0,2,1):
    for j in range(0,4,1):
        print(b[i][j],end="\t")
```

```
    print()
```

程序调试完成后，填写运行结果：

【实验Ⅱ-3】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_B3.py。

```
a=[[1,2,3,4],[5,6,7,8]]
b=[[0,0,0,0],[0,0,0,0]]
for i in range(0,2,1):
    for j in range(0,4,1):
        if j==0:
            b[i][3]=a[i][0]
        else:
            b[i][j-1]=a[i][j]
for i in range(0,2,1):
    for j in range(0,4,1):
        print(b[i][j],end="\t")
    print()
```

程序调试完成后，填写运行结果：

【实验Ⅱ-4】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_B4.py。

```
flag=0
a=[[1,2,3,4],[2,5,6,8],[3,6,7,9],[4,8,9,0]]
for i in range(0,4,1):
    for j in range(0,4,1) : print(a[i][j],end="\t")
    print()
for i in range(0,4,1):
    for j in range(0,i-1,1):
        if a[j][i]!=a[i][j]:
         flag=1
         break
if flag==1:
    print("No")
else:
    print("Yes")
```

程序调试完成后，填写运行结果：

【实验Ⅱ-5】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_B5. py。

```
a=[0,6,12,18,42,46,52,67,73]
x=52
n=len(a)
i=n//2+1
m=n//2
while m!=0:
    if x<a[i]:
        i=i-m//2-1
    elif x>a[i]:
        i=i+m//2+1
    else:
        break
    m=m//2
if m>0:
    print("The index is:",i)
else:
    print("Can't search")
```

程序调试完成后,填写运行结果:

【实验Ⅱ-6】 编程求出 100 以内的全部质数之和。

在 IDLE 编辑窗口中输入程序,并命名为 p12_B6. py。

【实验Ⅱ-7】 设定将 1000 元存入银行,年利率为 2.75%,编程将 5 年中的本息数据放入列表。

在 IDLE 编辑窗口中输入程序,并命名为 p12_B7py。

【实验Ⅱ-8】 编程将两个列表["A","B","C","D"]和["1","2","3","4"]合并为["A","1","B","2","C","3","D","4"]。

在 IDLE 编辑窗口中输入程序,并命名为 p12_B8. py。

【实验Ⅱ-9】 设定一个正整数,除以 2 得余数 1,除以 3 得余数 2,除以 5 得余数 4,除以 6 得余数 7,除以 7 得余数 0。编程找出在 10000 以内满足上述条件的全部正整数。

在 IDLE 编辑窗口中输入程序,并命名为 p12_B9. py。

在完成实验后,将 9 个源程序文件发送到教师邮箱中。

实验Ⅲ　函　　数

1. 实验目标

(1) 理解并掌握编程技术中的计算思维：分治、模块、递归和约简。

(2) 理解并掌握函数及其分类情况。

(3) 理解并实现函数定义与调用。

(4) 理解并掌握形式参数与实在参数的运用。

(5) 理解并掌握可变参数及其运用。

(6) 理解并掌握全局变量与局部变量的运用。

(7) 理解并实现 global 语句及其运用。

(8) 理解并实现 lambda 表达式。

(9) 理解并实现递归函数。

2. 实验环境

(1) Windows 7 操作系统及其以上版本。

(2) Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅲ-1】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C1.py。

```
def s(n):
    if n>1:
        return n+s(n-1)
    else:
        return 1
print(s(4))
```

程序调试完成后，填写运行结果：

【实验Ⅲ-2】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C2.py。

```
def incrementor(n):
    return lambda x : x +n
s=0
for i in range(1,101,1):
    m=incrementor(i)
    t=m(0)
    s=s+t
print("s=",s)
```

程序调试完成后，填写运行结果：

【实验Ⅲ-3】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C3. py。

```
def f(n):
    if n==1:
        y=6
    else:
        y=f(n-1)+2*n
    return y;
d=f(5);
print(d)
```

程序调试完成后，填写运行结果：

【实验Ⅲ-4】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C4. py。

```
def f(x,y):
    if x!=y:
        return (x+y)//2
    else:
        return x
a=4
b=5
c=6
print(f(2*a,f(b,c)))
```

程序调试完成后，填写运行结果：

【实验Ⅲ-5】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C5. py。

```
a=12
b=16
c=40
def large(a,b):
    if a>b:
        c=a
    else:
        c=b
    return c
a=20
print(large(a,b))
```

程序调试完成后，填写运行结果：

【实验Ⅲ-6】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C6.py。

```
def sum(n):
    s=0
    for i in range(1,n+1,1) :
        if i%2==0 : s=s+i
    return s
number=int(input("n="))
print(sum(number))
```

输入数据 10，程序调试完成后，填写运行结果：

【实验Ⅲ-7】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C7.py。

```
def f(str):
    count=0
    for i in range(0,len(str)):
        if str[i]=="i" : count=count+1
    return count
s="This is a string."
print(f(s))
```

程序调试完成后，填写运行结果：

【实验Ⅲ-8】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_C8.py。

```
a=10
def s():
    a=10
    a=a+10
    print("a=",a)
s()
for i in (1,3,1) : print("a=",a)
s()
```

程序调试完成后，填写运行结果：

【实验Ⅲ-9】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_C9.py。

```
def f(n):
    if n==1:
        s=1
    elif n==2:
        s=2
    else:
        s=n+f(n-1)
    return s
print(f(4))
```

程序调试完成后,填写运行结果:

【实验Ⅲ-10】 参照例 4-48 编写函数生成图Ⅲ-1 所示的 7 阶幻方,并在主程序中调用。

30	39	48	1	10	19	28
37	47	7	9	18	27	29
46	6	8	17	26	35	37
5	14	16	25	34	36	45
13	15	24	33	42	44	4
21	23	32	41	43	3	12
22	31	40	49	2	11	20

图Ⅲ-1　7 阶幻方

在 IDLE 编辑窗口中输入程序,并命名为 p12_C10.py。

【实验Ⅲ-11】 编写函数将正整数 n 转换为二进制形式并调用。(提示:使用除二取余算法,并用列表存放全部二进制的数位。例如输入 1234,则输出 010011010010。)

在 IDLE 编辑窗口中输入程序,并命名为 p12_C11.py。

在完成实验后,将 11 个源程序文件发送到教师邮箱中。

实验Ⅳ　数 据 文 件

1. 实验目标

(1) 理解并掌握数据文件的概念。

(2) 理解并掌握标准输入文件与标准输出文件的概念。

(3) 理解并掌握 ASCII 文件与二进制文件的概念。

(4) 理解并掌握顺序存取文件与随机存取文件的概念。

(5) 理解并实现打开文件和关闭文件。

(6) 理解并实现文本文件的读取操作和写入操作。

(7) 理解并实现二进制文件的读取操作和写入操作。

2. 实验环境

(1) Windows 7 操作系统及其以上版本。

(2) Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅳ-1】　在 IDLE 编辑窗口中输入如下程序,并命名为 p12_D1.py。

```
f=open("d:\\Python36\\chc\\ex01.txt","r")
line="begin"
while line!="":
    line=f.read()
    print(line)
f.close()
```

运行程序前,在文件 ex01.txt 中存放 3 个字符串:

```
Python Programming
Computational Thinking Perspective
TsingHua University Press
```

程序调试完成后,应该能够验证运行结果是否正确。

【实验Ⅳ-2】　在 IDLE 编辑窗口中输入如下程序,并命名为 p12_D2.py。

```
f=open("d:\\Python36\\chc\\ex02.txt ","w")
f.write("Python 程序设计\n ")
f.write("计算思维视角\n ")
f.write("清华大学出版社\n ")
f.close()
```

程序调试完成后,应该自行查看 ex02.txt 文件以便验证运行结果是否正确。

【实验Ⅳ-3】　在文件 ex03.txt 中分行保存如下字符串:"Java" "C" "C++" "C#" "Python"和"JavaScript",编程从数据文件中读取字符串并分别计算串长。

在 IDLE 编辑窗口中输入程序，并命名为 p12_D3. py。

【实验Ⅳ-4】 编程生成特殊斐波那契数（前 3 个数分别为 1，2，3，其后的任何数均是前 3 个数之和），并按下面格式写入到文件 ex04. txt 中。

6	11	20	37	68
125	230	423	778	1431
2632	4841	8904	16377	30122
55403	101902	187427	344732	634061

在 IDLE 编辑窗口中输入程序，并命名为 p12_D4. py。

【实验Ⅳ-5】 编程计算 1！～12！，并将结果分 12 行写入文件 ex05. txt 中。

在 IDLE 编辑窗口中输入程序，并命名为 p12_D5. py。

【实验Ⅳ-6】 在文件 ex06. txt 中存放字符串"Python"，编程逐个读取字符并分 6 行显示。

在 IDLE 编辑窗口中输入程序，并命名为 p12_D6. py。

【实验Ⅳ-7】 编程遍历文件对象并从中读取文件中的每一行，文件 ex07. txt 中的内容如下所示：

```
Python 是易于学习的编程语言
Python 是功能强大的编程语言
Python 是解释性质的编程语言
Python 具有面向对象编程方法
```

在 IDLE 编辑窗口中输入程序，并命名为 p12_D7. py。

【实验Ⅳ-8】 在文件 ex08. txt 中写入如下文本：

> Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

编程分别统计其中的字母、空格和其余符号的数量。

在 IDLE 编辑窗口中输入程序，并命名为 p12_D8. py。

在完成实验后，将 8 个源程序文件和全部数据文件发送到教师邮箱中。

实验Ⅴ　面向对象编程

1. 实验目的

(1) 理解并掌握编程技术中的计算思维：抽象、对象、封装、多态和继承。

(2) 理解并掌握对象及其组成。

(3) 理解并掌握类及其组成。

(4) 理解并实现单继承与多继承。

(5) 理解并实现多态。

(6) 理解并实现运算符重载。

(7) 理解并实现类的定义和引用。

(8) 理解并实现构造函数和析构函数。

(9) 理解并实现类属性和实例属性。

(10) 理解并实现私有成员与公有成员。

(11) 理解并实现公有方法和私有方法。

(12) 理解并实现方法重写。

2. 实验环境

(1) Windows 7 操作系统及其以上版本。

(2) Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅴ-1】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_E1.py。

```
class Student_class:
    xh=2017123456789
    xm="王小华"
    cs="四川大学计算机学院"
    def display(self):
        print("学号：",self.xh,end="\t")
        print("姓名：",self.xm,end="\t")
        print("学院：",self.cs)
stu=Student_class()
print("第 1 个类变量：",stu.xh)
print("第 2 个类变量：",stu.xm)
print("第 3 个类变量：",stu.cs)
stu.display()
```

程序调试完成后，填写运行结果：

【实验Ⅴ-2】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_E2.py。

```
class Circle_class:
    radius=10
    def computation(self):
        print("类变量: ",self.radius)
        print("圆周长: ",2*3.14*self.radius)
        print("圆面积: ",3.14*self.radius*self.radius)
        print("球体积: ",4*3.14*self.radius*self.radius*self.radius/3)
cir=Circle_class()
print(cir.radius)
cir.computation()
```

程序调试完成后,填写运行结果:

【实验Ⅴ-3】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_E3.py。

```
class Book_class:
    price=30
    def __init__(self,book_name,book_price):
        self.sm=book_name
        self.__pr=book_price
books1=Book_class("Python: ",38)
print("类变量: ",books1.price)
books2=Book_class("JavaScript: ",50)
print("类变量: ",books1.price)
print("第 1 本图书: ",books1.sm,end="\t")
print("定价: ",books1._Book_class__pr)
print("第 2 本图书: ",books2.sm,end="\t")
print("定价: ",books2._Book_class__pr)
```

程序调试完成后,填写运行结果:

【实验Ⅴ-4】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_E4.py。

```
class Books:
    price=20
    def __init__(self):
        self.__color="颜色:白色"
        self.__citys="城市:北京"
    def __displayColor(self):
        print(self.__color)
    def __displaycitys(self):
      print(self.__citys)
    def display(self):
        self.__displayColor()
        self.__displaycitys()
bk=Books()
bk.display()
```

程序调试完成后,填写运行结果:

【实验Ⅴ-5】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_E5.py。

```
class Parent_class:
    Parent_classAttr=40
    def __init__(self):
        print ("这是编程技术大类")
    def Parent_classMethod(self):
        print("显示编程技术大类")
    def setAttr(self,attr):
        Parent_class.Parent_classAttr=attr
    def getAttr(self):
      print("编程技术大类的定价:",Parent_class.Parent_classAttr)
class Child_class(Parent_class):
    def __init__(self):
```

```
        print("这是 Python 编程子类")
    def Child_classMethod(self):
        print("显示 Python 编程子类")
exp=Child_class()
exp.Child_classMethod()
exp.Parent_classMethod()
exp.setAttr(36)
exp.getAttr()
```

程序调试完成后，填写运行结果：

【实验Ⅴ-6】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_E6.py。

```
class Books:
    def list(self):
        print("这是编程技术大类")
class bk1(Books):
    def list(self):
        print("这是 Python 编程子类")
class bk2(Books):
    def list(self):
      print("这是 JavaScript 编程子类")
c=bk2()
c.list()
```

程序调试完成后，填写运行结果：

【实验Ⅴ-7】 设定汽车信息为车型、排量、品牌、制造商和定价，编程定义并访问含类变量和成员方法的类，其中成员方法能够显示汽车信息。

在 IDLE 编辑窗口中输入程序，并命名为 p12_E7.py。

在完成实验后，将 7 个源程序文件发送到教师邮箱中。

实验Ⅵ 图形界面设计

1. 实验目标

(1) 理解并掌握图形界面窗口的创建模式。

(2) 理解并实现布局管理。

(3) 理解并实现 Tkinter 模块中的主要组件,包括按钮、输入框、框架、标签、列表框、菜单、滚动条、文本框、滑动杆、面板、对话框和消息框。

(4) 理解并掌握事件处理。

(5) 理解并实现事件的对象绑定。

(6) 理解并实现事件的标识绑定。

2. 实验环境

(1) Windows 7 操作系统及其以上版本。

(2) Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅵ-1】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_F1.py。

```
from tkinter import *
root=Tk()
root.title("由标签、文本输入框和按钮组成的窗口")
root["width"]=200
root["height"]=80
Label(root,text="账号",width=8).place(x=1,y=1)
Entry(root,width=24).place(x=45,y=1)
Label(root,text="口令",width=8).place(x=1,y=20)
Entry(root,width=24, show="*").place(x=45,y=20)
Button(root,text="注册",width=6).place(x=48,y=60)
Button(root,text="放弃",width=6).place(x=164,y=60)
```

程序调试完成后,要求将运行结果截图并存入"实验 6.doc"文档中。

【实验Ⅵ-2】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_F2.py。

```
import tkinter as tk
def  btnHelloClicked():
    degree=float(entryCd.get())
    labelHello.config(text="%.2f 度数值=%.2f 弧度值" %(degree,degree*3.14/180))
root=tk.Tk()
root.title("弧度转换")
labelHello=tk.Label(root,text="转换度数值为弧度值",height=5,width=20,fg="blue")
labelHello.pack()
entryCd=tk.Entry(root)
```

```
entryCd.pack()
btnCal=tk.Button(root,text="计算弧度",command=btnHelloClicked)
btnCal.pack()
root.mainloop()
```

程序调试完成后,要求将运行结果截图并存入"实验 6.doc"文档中。

【实验Ⅵ-3】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_F3.py。

```
from tkinter import *
root=Tk()
root.title("由列表框、按钮和鼠标事件组成的窗口")
def callbutton1():
    for i in listb1.curselection():
        listb2.insert(0,listb1.get(i))
def callbutton2():
    for i in listb2.curselection():
        listb2.delete(i)
lst=["monday","tuesday","thursday","wednesday","friday","saturday","sunday"]
listb1=Listbox(root)
listb2=Listbox(root)
for item in lst:
    listb1.insert(0,item)
listb1.grid(row=0,column=0,rowspan=2)
b1=Button(root,text="添加>>",command=callbutton1,width=20)
b2=Button(root,text="删除<<",command=callbutton2,width=20)
b1.grid(row=0,column=1,rowspan=2)
b2.grid(row=1,column=1,rowspan=2)
listb2.grid(row=0,column=2,rowspan=2)
root.mainloop()
```

程序调试完成后,要求将运行结果截图并存入"实验 6.doc"文档中。

【实验Ⅵ-4】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_F4.py。

```
import tkinter
root=tkinter.Tk()
root.title("由单选按钮组成的窗口")
rt=tkinter.StringVar()
rt.set("3")
radio=tkinter.Radiobutton(root,variable=rt,value="1",text="自动机理论")
radio.pack()
radio=tkinter.Radiobutton(root,variable=rt,value="2",text="形式语言理论")
radio.pack()
radio=tkinter.Radiobutton(root,variable=rt,value="3",text="可计算性理论")
radio.pack()
radio=tkinter.Radiobutton(root,variable=rt,value="4",text="计算复杂性理论")
radio.pack()
radio=tkinter.Radiobutton(root,variable=rt,value="5",text="算法理论")
```

```
radio.pack()
radio=tkinter.Radiobutton(root,variable=rt,value="6",text="计算机数学")
radio.pack()
root.mainloop()
print(r.get())
```

程序调试完成后，要求将运行结果截图并存入"实验 6. doc"文档中。

【实验Ⅵ-5】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_F5. py。

```
from tkinter import *
root=Tk()
root.title("由框架和按钮组成的窗口")
f1=Frame(root)
f1.pack()
f2=Frame(root)
f2.pack()
redbutton=Button(f1,text="红字",fg="red")
redbutton.pack( side=LEFT)
brownbutton=Button(f1,text="棕字",fg="brown")
brownbutton.pack( side=LEFT )
bluebutton=Button(f1,text="蓝字",fg="blue")
bluebutton.pack( side=LEFT )
blackbutton=Button(f2,text="黑字",fg="black")
blackbutton.pack()
root.mainloop()
```

程序调试完成后，要求将运行结果截图并存入"实验 6. doc"文档中。

【实验Ⅵ-6】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_F6. py。

```
import tkinter
from tkinter import simpledialog
def inputInt():
    r=simpledialog.askinteger("Python Tkinter","请输入一个整数")
    print(r)
def inputFloat():
    r=simpledialog.askfloat("Python Tkinter","请输入一个实数")
    print(r)
root=tkinter.Tk()
root.title("由简单对话框和按钮组成的窗口")
btn1=tkinter.Button(root,text="输入整数",command=inputInt)
btn2=tkinter.Button(root,text="输入实数",command=inputFloat)
btn1.pack(side="left")
btn2.pack(side="left")
root.mainloop()
```

程序调试完成后，要求将运行结果截图并存入"实验 6. doc"文档中。

【实验Ⅵ-7】 单击按钮(Button)后在消息框(messagebox)中显示：欢迎学习 Python

程序设计。

在 IDLE 编辑窗口中输入程序，并命名为 p12_F7.py。

【实验Ⅵ-8】 编程：设置由 6 本图书信息构成的单选按钮窗口，且初选状态是“李开复：做最好的自己”。

余秋雨：山居笔记

李开复：做最好的自己

周国平：各自的朝圣路

王小波：我的精神家园

王小慧：我的视觉日记

莫罗阿：人生五大问题

在 IDLE 编辑窗口中输入程序，并命名为 p12_F8.py。

在完成实验后，将文档“实验 6.doc”和 8 个源程序文件发送到教师邮箱中。

实验Ⅶ　绘 制 曲 线

1. 实验目标

(1) 理解并掌握 Canvas(画布)及其创建 Canvas 对象的方法。

(2) 理解并掌握屏幕坐标的图形显示方式。

(3) 理解并掌握 Canvas 画布对象内含的图形绘制函数和控制图形函数。

(4) 理解并实现调用 IDLE 环境中的 Turtle Demo 演示程序的方法。

(5) 理解并掌握海龟绘图的基本属性、运动控制命令和控制画笔命令。

(6) 理解并掌握分形图形的构造方法。

(7) 理解并实现通过元组表示字体的方法。

(8) 理解并实现通过 Font 对象表示字体的方法。

2. 实验环境

(1) Windows 7 操作系统及其以上版本。

(2) Python 3.6 及其以上版本。

3. 实验过程

【实验Ⅶ-1】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_G1.py。

```
from tkinter import *
root=Tk()
root.title("由两个矩形构成的窗口")
cv=Canvas(root,bg='white',width=320,height=140)
cv.create_rectangle(10,10,120,120,width=2,fill="blue")
cv.create_rectangle(140,10,320,120,width=3,outline="blue")
cv.pack()
root.mainloop()
```

程序调试完成后,要求将运行结果截图并存入"实验 7.doc"文档中。

【实验Ⅶ-2】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_G2.py。

```
from tkinter import *
root=Tk()
root.title("由 3 个多边形及填充颜色构成的窗口")
cv=Canvas(root,bg="gray",width=240,height=120)
cv.create_polygon(35,10,10,60,60,60,outline="red",fill="white")
cv.create_polygon(70,10,120,10,120,60,outline="red",width=2)
cv.create_polygon(130,10,180,10,180,60,130,60)
cv.pack()
root.mainloop()
```

程序调试完成后,要求将运行结果截图并存入"实验 7.doc"文档中。

【实验Ⅶ-3】 在 IDLE 编辑窗口中输入如下程序,并命名为 p12_G3.py。

```
from tkinter import *
root=Tk()
root.title("由椭圆、圆和填充颜色构成的窗口")
cv=Canvas(root,bg="gray",width=200,height=100)
cv.create_oval(10,10,100,50,outline="red",fill="white",width=2)
cv.create_oval(100,10,190,100,outline="blue",fill="white",width=2)
cv.pack()
root.mainloop()
```

程序调试完成后，要求将运行结果截图并存入“实验 7. doc”文档中。

【实验Ⅶ-4】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_G4. py。

```
from tkinter import *
root=Tk()
root.title("由两个矩形和缩放图形操作构成的窗口")
cv=Canvas(root,bg="white",width=240,height=180)
rt1=cv.create_rectangle(10,10,110,110,outline="blue",width=2)
rt2=cv.create_rectangle(20,20,120,120,outline="black",width=2)
cv.scale(rt1,10,10,1.6,1)
cv.pack()
root.mainloop()
```

程序调试完成后，要求将运行结果截图并存入“实验 7. doc”文档中。

【实验Ⅶ-5】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_G5. py。

```
import turtle
import time
turtle.color("black")
turtle.pensize(3)
turtle.speed(3)
turtle.goto(0,0)
for i in range(4):
    turtle.forward(100)
    turtle.right(90)
turtle.up()
turtle.goto(-150,-120)
turtle.color("red")
turtle.write("Over")
time.sleep(3)
```

程序调试完成后，要求将运行结果截图并存入“实验 7. doc”文档中。

【实验Ⅶ-6】 在 IDLE 编辑窗口中输入如下程序，并命名为 p12_G6. py。

```
from tkinter import *
root=Tk()
root.title("由五种显示字体构成的窗口")
fnt=(("MS Serif",),"Symbol",("Times New Roman",),"Fixdsys","Verdana")
```

```
for ft in fnt:
    Label(root,text="Representing font by tuple",font=ft).grid()
root.mainloop()
```

程序调试完成后，要求将运行结果截图并存入“实验 7. doc”文档中。

【实验Ⅶ-7】 调用 create_polygon()函数分别绘制一个正方形、一个矩形和一个平行四边形。

在 IDLE 编辑窗口中输入程序，并命名为 p12_G7. py。

【实验Ⅶ-8】 调用 create_arc()函数分别绘制 3 个平行的相邻椭圆。

在 IDLE 编辑窗口中输入如下程序，并命名为 p12_G8. py。

【实验Ⅶ-9】 调用 create_bitmap()函数分别显示 3 个平行的位图。

在 IDLE 编辑窗口中输入程序，并命名为 p12_G9. py。

在完成实验后，将文档“实验 7. doc”和 9 个源程序文件发送到教师邮箱中。

参 考 文 献

[1] 陆朝俊. 程序设计思想与方法——问题求解中的计算思维[M]. 北京：高等教育出版社，2013.
[2] 沙行勉. 计算机科学导论——以 Python 为舟 [M]. 2 版. 北京：清华大学出版社，2016.
[3] 夏敏捷，等. Python 程序设计——从基础到开发[M]. 北京：清华大学出版社，2017.
[4] 吴萍，等. 算法与程序设计基础(Python 版)[M]. 北京：清华大学出版社，2015.
[5] 江红，等. Python 程序设计与算法基础教程[M]. 北京：清华大学出版社，2017.
[6] 董付国. Python 程序设计[M]. 2 版. 北京：清华大学出版社，2016.
[7] 嵩天，等. Python 语言程序设计基础[M]. 2 版. 北京：高等教育出版社，2017.
[8] 刘卫国. Python 语言程序设计[M]. 北京：电子工业出版社，2016.